PRACTICAL APPLICATION OF REXX IN THE UNIX ENVIRONMENT

ED SPIRE
THE WORKSTATION GROUP

PRACTICAL APPLICATION OF REXX IN THE UNIX ENVIRONMENT

1. COMMERCIAL USERS MIGRATING TO UNIX.

2. UNIX HAS A LARGE LEARNING CURVE.

3. REXX CAN EASE THE TRANSITION BY PROVIDING A FAMILIAR FACILITY.

4. REXX BRINGS A NEW LEVEL OF FUNCTIONALITY TO UNIX.

Types of REXX applications in Unix:

1. Unix command macros
2. Macros for other utilities which support REXX directly
3. General purpose programming in REXX
4. Embedded REXX applications

UNIX COMMAND MACROS:

1. RECORD RESEARCHED TECHNIQUES FOR FUTURE USE
2. SIMPLIFY UNIX COMMAND SYNTAX
3. AUTOMATE REPEATED USAGE OF RELATED UNIX COMMAND SEQUENCES
4. PROVIDE ACCESS TO FEATURES THAT ARE OTHERWISE DIFFICULT TO USE
5. EXTEND THE OPERATING SYSTEM'S FACILITIES

UNIX COMMAND MACROS: RECORD RESEARCHED TECHNIQUES FOR FUTURE USE

INSTEAD OF	CAT <FILE> | RSH SCOTTY LPR
ALLOW	RLP <FILE>

```
#!/usr/local/bin/rxx
/*
 * rlp - print on a printer on another machine
 *
 * rlp filename machine traceopt
 *
 *      filename is the name of the file to be printed.
 *      machine is the machine that has the desired printer (defaults
 *              to scotty)
 *      traceopt is a rexx trace option, defaults to no tracing.
 */
parse arg fn machine traceopt
trace value traceopt
if machine="" then machine="scotty"
"cat" fn "| rsh" machine "lpr"
```

UNIX COMMAND MACROS: SIMPLIFY UNIX COMMAND SYNTAX

INSTEAD OF FIND /USR -NAME <THINGY> -PRINT

ALLOW FI <THINGY>

```
#!/usr/local/bin/rxx
/*
 * fi - run find on just /usr, where everything is anyway.
 *
 * this helps you not run find on the root, which would go out and
 * look through all your nfs mounts.  It also helps you not have to
 * remember the find command's syntax...
 */
parse arg name
"find /usr -name" name "-print"
```

UNIX COMMAND MACROS: AUTOMATE REPEATED SEQUENCES

INSTEAD OF	PS -U <USERID> (VISUALLY LOOK FOR A LINE REFERRING TO <PGM> AND REMEMBER ITS <PROCESS ID>) DBX -A <PROCESS-ID>
ALLOW	DBXW <PGM>

```
#!/usr/local/bin/rxx
/*
 * dbxw - run dbx on the program running in another window.
 * This is useful when the program in the other window is a curses
 * application and the dbx output would mess up its "screen" display.
 *
 * dbx programname
 *
 * will run a ps -u userid and look for a process running programname,
 * and then dbx -a processid.
 *
 * Note that you should probably cd to the directory where the program
 * resides before you dbxw.
 */
parse arg programname traceopt
trace value traceopt
call popen "ps -u" userid() "| grep" programname
select
  when queued()=0
   then say "can't find" programname
  when queued()=1
   then do
        parse pull processid .
        "dbx -a" processid
        end
  when queued()>1
   then say "more than one" programname "running!"
  end
```

UNIX COMMAND MACROS: ACCESS TO OTHERWISE HARD TO USE FEATURES

INSTEAD OF	?????? (TO SET <TITLE> AS THE TITLE OF AN X WINDOW AND ITS ICON)
ALLOW	SETNAMES <TITLE>

```
#!/usr/local/bin/rxx
/*
 * setnames - change the name associated with an X window.
 *
 *    The name of the window or its icon can be changed by sending
 *    a specific escape sequence to the terminal window...
 */
escape = x2c("1b")
parse arg name                                    /* charout avoids the */
call charout , escape||"]l"||name||escape  /* unwanted 'cr' that */
call charout , escape||"]L"||name||escape  /* lineout would send */
```

Typical usage of setnames:

```
#!/usr/local/bin/rxx
/*
 * rl - rlogin to another system, changing the names in the cterm window
 *      and icon to reflect that system's name
 */
parse arg system                                        /* who to rlogin to */
"setnames" system                                        /* put his name up */
"rlogin" system                                            /* rlogin to him */
call popen "hostname"                                        /* who are we? */
parse pull hostname
"setnames" hostname                          /* restore this system's name */
```

UNIX COMMAND MACROS: SIMPLIFY UNIX COMMAND SYNTAX (LARGE SCALE)

REXX UTILITY TO PARSE LARGE NUMBERS OF OPERANDS

(CALLING SEQUENCE SHOWN)

```
/******************** PARSE SEQUENCE PATTERN ****************************/
/* MODIFY THE THIRD LINE AS YOUR "PROTOTYPE" SHOWING PARMS AND DFLTS  */
/******************** START OF PARSE SEQUENCE ****************************/
parse arg a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18
interpret cparse(,
"p1 p2(*) ( nk1 nk2 k1(k1v) k2() abc",
||" )" a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18)
/******************** END OF PARSE SEQUENCE ****************************/
```

UNIX COMMAND MACROS: SIMPLIFY UNIX COMMAND SYNTAX (LARGE SCALE)

SOME UNIX COMMANDS (ESPECIALLY THOSE ASSOCIATED WITH THE X WINDOWS SYSTEM) CAN HAVE LOTS OF OPERANDS...

```
xterm [ -ah] [ -ar] [ -b NumberPixels] [ -bd Color] [ -bg Color]
      [ -bw NumberPixels] [ -ccCharRange:Value[,...]]
      [ -cr Color] [ -cu] [ -display Name:Number] [ -dw]
      [ -fb Font] [ -fg Color] [ -fn Font] [ -fr Font]
      [ -fullcursor] [ -geometry Geometry] [ #Geometry] [ -help]
      [ -i] [ -ib File] [ -j] [ -keywords] [ -lang Language] [ -l]
      [ -leftscroll] [ -lf File] [ -ls] [ -mb] [ -mc Number]
      [ -ms Color] [ -n IconName] [ -name Application]
      [ -nb Number] [ -po Number] [ -ps] [ -reduced] [ -rv]
      [ -rw] [ -s] [ -sb] [ -sf] [ -si] [ -sk]
      [ -sl NumberLines] [ -sn] [ -st] [ -suppress] [ -T Title]
      [ -ti] [ -tm String] [ -tn TerminalName] [ -ut]
      [ -v] [ -vb] [ -W] [ -xrm String] [ -132] [ -e Command]
```

Examples

The following example can be used to create an xterm, specifying the size and location of the window, using a font other than the default, and also specifying the foreground color to be used of the text. It then runs a command in that window.

```
xterm -geometry 20x10+0+175 -fn Bld14.500 -fg DarkTurquiose
                            -e /tmp/banner_cmd &
```

UNIX COMMAND MACROS: SIMPLIFY UNIX COMMAND SYNTAX (LARGE SCALE)

SIMPLIFIED XTERM, WITH NEW DEFAULTS AND EASY SPECIFICATION OPERANDS

```
#!/usr/local/bin/rxx
/*
 * xt - start an xterm window
 *
 * The positional parms comprise a Unix command that is to be run in this
 * window.  If none is specified, then your normal shell is run instead.
 *
 * optional parms:
 *   l #  - number of lines (default 25)
 *   x #  - x component of window location
 *   y #  - y component of window location
 *   i    - if present, window starts as an icon.
 *   fr # - reduced screen font size (default 14, the typical default
 *          for normal aixterm windows.)
 *   fn # - normal screen font size (default 10, small than typical for
 *          aixterm windows).
 *             The above two default settings make for normally small
 *             windows, which can be temporarily enlarged back to their
 *             traditional size by selecting "reduced" from the alt-
 *             left button menu.
 *   s    - if present, xterm is run synchronously.
 *   test - if present, the options line is shown on the screen and
 *          aixterm invocation is supressed.
 */
/******************** PARSE SEQUENCE PATTERN ****************************/
/* MODIFY THE THIRD LINE AS YOUR "PROTOTYPE" SHOWING PARMS AND DFLTS  */
/******************** START OF PARSE SEQUENCE ***************************/
parse arg a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18
interpret cparse(,
"c1() c2() c3() c4() c5() c6() ( l(25) x() y() i fr(14) fn(10) test s",
||" )" a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18)
/******************** END OF PARSE SEQUENCE *****************************/
cmd=c1 c2 c3 c4 c5 c6                          /* build the desired command */
if cmd="" then command = ""                 /* build the required xterm option */
         else command = "-e" cmd
if s="" then amp="&"  /* build the required background execution option */
        else amp=""
options="-fullcursor -sb -sl 999 -ar -ls"  /* initial xterm options set */
options=options "-geometry 80x"l                  /* number of lines option*/
if x<>"" | y<>"" then do                         /* if position specified, */
     if x="" then x=0                        /* fill in remaining defaults */
     if y="" then y=0
     options=options||"+"||x||"+"||y                  /* position option */
     end
options=options "-fn Rom"||fn||".500"                 /* normal font option*/
options=options "-fb Rom"||fn||".500"                   /* bold font option*/
options=options "-fr Rom"||fr||".500"               /* reduced font option */
if i="i" then options=options "-i"      /* if req, then start as an icon */
call chdir("/u/ets")                             /* back to home directory */
if test=""                                          /* show it or do it */
then "/usr/lpp/X11/bin/aixterm" options command amp
else say options command amp
```

UNIX COMMAND MACROS: SIMPLIFY UNIX COMMAND SYNTAX (LARGE SCALE)

EXAMPLE USAGE OF XT:

```
#!/usr/local/bin/rxx
/*
 * xi - initialize xterm environment
 *
 * this just creates my standard set of windows, with their normal
 * positions, but leaves them all as icons at first.
 */
"xt            ] x 0   y 0   i l 64"              /* large primary window */
"xt            ] x 680 y 0   i"                   /* 1st alternate window */
"xt            ] x 680 y 395 i"                   /* 2nd alternate window */
"xt rl wrkgrp  ] x 0   y 390 i"     /* window on wrkgrp (Sun-3) */
"xt rl drwho   ] x 575 y 630 i"     /* window on drwho (Sparc) */
"xt rl scotty  ] x 600 y 590 i"     /* window on scotty (SCO/Unix) */
"xt rl orac    ] x 0   y 545 i"     /* window on orac (HP-9000/300 HP-UX */
"xt rl worf    ] x 0   y 545 i"     /* window on orac (HP-9000/300 Domain-OS*
/*
 * orac and worf are in the same spot, since they are the same machine,
 * and only one will be up at a time.  The other will die quitely
 * after a few attempts to rlogin
 */
```

UNIX COMMAND MACROS: EXTEND UNIX FACILITIES

INSTEAD OF UNDERSTANDING YOUR LOCAL NFS NETWORK AND HOW TO TRANSLATE LOCAL FILENAMES ON ONE SYSTEM TO THE CORRESPONDING LOCAL FILENAME ON YOUR SYSTEM...

ALLOW FILENAME SYNTAX OF

NODE:/LOCAL/FILE/NAME

THROUGHOUT A SET OF UNIFORM UTILITIES.

Filesystem	Total KB	free	%used	iused	%iused	Mounted on
/dev/hd4	49152	11356	76%	1146	9%	/
/dev/hd2	225280	37716	83%	6367	11%	/usr
/dev/hd3	32768	31700	3%	26	0%	/tmp
/dev/hd1	249856	223444	10%	1292	2%	/u
wrkgrp:/home	47946	5832	87%	-	-	/sun
wrkgrp:/usr	213313	18127	91%	-	-	/sunusr
wrkgrp:/	7608	2635	65%	-	-	/sunroot
drwho:/home	326519	66998	79%	-	-	/drwho
scotty:/	99037	4694	95%	-	-	/odt
drwho:/usr	183439	47344	74%	-	-	/whousr

UNIX COMMAND MACROS: EXTEND UNIX FACILITIES

INSTEAD OF UNDERSTANDING YOUR LOCAL NFS NETWORK AND HOW TO TRANSLATE LOCAL FILENAMES ON ONE SYSTEM TO THE CORRESPONDING LOCAL FILENAME ON YOUR SYSTEM...

ALLOW FILENAME SYNTAX OF

NODE:/LOCAL/FILE/NAME

THROUGHOUT A SET OF UNIFORM UTILITIES.

```
/*
 * fn filename
 *
 * fn accepts a filename in a system independent form, and generates
 * a local filename which will provide (probably NFS) access to the
 * desired file.
 *
 * filename has the form
 *
 *   host:/filename/on/that.host
 *
 * note:  if host: is omitted, then no translation is done, assuming
 * that a local filename was really specified in the first place
 */
parse arg host ':' file
call popen 'hostname'
parse pull currenthost
select
when file=""                /* if no "host:", parse will have put it all */
then o=host                                  /* in host, and file will be null. */
when host=currenthost /* if explicitly referring to a file on this host */
then o=file                                          /* just use that file name */
otherwise do
 call popen 'df'
 lm=""                    /* will become the saved df line that matches. */
 do while queued()>0
  parse pull l
  parse var l dfhost ':' dfhostdir dfjunk '/' dflocaldir
  if length(dfhostdir)>0 ,                    /* weeds out header and locals */
      & dfhost==host                               /* weeds out other systems */
  then do
   if dfhostdir=="/"                                    /* for root file system */
   then lm=l                    /* save this line in case we find no other */
   else if left(file,length(dfhostdir))==dfhostdir       /* right line? */
   then do
    lm=l
    leave
    end
   end
  end
 if length(lm)>0                                       /* if we found something, */
 then do
  parse var lm dfhost ':' dfhostdir dfjunk '/' dflocaldir
  if dfhostdir^=='/'                                   /* if not root filesystem, */
  then file=right(file,length(file)-length(dfhostdir))  /* trim rmt dir */
  o='/'||dflocaldir||file                              /* add correct local dir */
  end
 else do                  /* if we found nothing, fail with error message */
      say 'sorry, no path from here to' host':'file
      return                          /* return with no value is a failure */
      end
 end
end
return o                                                /* non-failure return */
```

UNIX COMMAND MACROS: EXTEND UNIX FACILITIES

INSTEAD OF UNDERSTANDING YOUR LOCAL NFS NETWORK AND HOW TO TRANSLATE LOCAL FILENAMES ON ONE SYSTEM TO THE CORRESPONDING LOCAL FILENAME ON YOUR SYSTEM...

ALLOW FILENAME SYNTAX OF

NODE:/LOCAL/FILE/NAME

THROUGHOUT A SET OF UNIFORM UTILITIES.

Typical usages of fn allow the user to access files on other systems without knowing the details of the NFS links that connect these systems.

```
#!/usr/local/bin/rxx
/*
 * nfl - invoke flist with system independent filename
 */
parse arg dir
ldir=fn(dir)
say 'flist' ldir
'flist' ldir
```

```
#!/usr/local/bin/rxx
/*
 * nxe - invoke xedit with a system independent filename
 */
'xe' fn(arg(1))
```

Macros for other utilities that support REXX

1. Record researched techniques for future use
2. Extend the features of that utility
3. Integrate the utility with other Unix operations

MACROS FOR OTHER UTILITIES: RECORD RESEARCHED TECHNIQUES

TO PRINT PART OF THE CURRENT XEDIT FILE,

INSTEAD OF	!RM TEMP PUT <TARGET> TEMP !RLP TEMP
ALLOW	RLP <TARGET>

```
/*
 * rlp.xedit - an xedit macro to print part of an xedit file
 *
 * rlp target
 *
 *   is the same as
 *
 *        !rm temp
 *        put target temp
 *        !rlp temp
 *
 */
parse arg target
address unix  'rm temp'
address xedit 'put' target 'temp'
address unix  'rlp temp'
```

MACROS FOR OTHER UTILITIES: EXTEND FEATURES

PROVIDE PARAGRAPH REFORM CAPABILITIES IN XEDIT

```
/* FLOW MACRO.
This macro aligns two or more lines of a text-type file
being edited (such as a NOTE). It tries to place as many
words as possible on a line, within the right margin
defined by XEDIT SET TRUNC.

USE:

    FLOW <target>

where <target> is a standard Xedit target defining the first line
not to be flowed. Typically, the alignment process will result in
there being fewer lines in the block than there were before alignment.
This will not always be true.

UNIQUE CAPABILITY OF THIS PROGRAM:

This macro can, unlike other parts of XEDIT, shorten lines. If you
SET TRUNC to a value shorter than some of the lines in your file,
they will be handled correctly by this macro. Elsewhere in XEDIT,
results are unpredictable and will likely involve data loss.

MODIFICATION HISTORY:

11/16/86 - Roger Deschner - Original version
01/02/88 - Roger Deschner - Replace call to "JOIN", for performance
02/14/88 - Roger Deschner - Allow lines to be shortened; use PUTD
10/24/89 - Roger Deschner - Protect from LINEND character
10/04/90 - Roger Deschner - Changed to FLOW; moved to RS/6000
*/

/* Do it to it */

doit:
PARSE ARG targ
```

```
doit:
PARSE ARG targ
tempfile = 'JJ.TEMP'
ADDRESS UNIX 'rm -f' tempfile
'PUTD' targ tempfile
'UP 1'
'EXTRACT /TRUNC'

rotbuf = ''  /* initialize rotating buffer */
DO FOREVER
  IF (LINES(tempfile) = 0) THEN LEAVE    /* EOF? */
  ibuf = LINEIN(tempfile)
  IF (SUBSTR(ibuf,1,1) = ' ') THEN DO /* Paragraph break, either kind */
    IF (rotbuf ^= ' ') THEN DO    /* Anything left in buffer? */
      'INPUT' rotbuf    /* put it out */
      rotbuf = ''
    END
  END
  IF (ibuf = ' ') THEN 'INPUT   '    /* Blank line */
  ELSE DO                          /* duit tuit */
    /* concatenate the new stuff */
    IF (rotbuf = '') THEN rotbuf = STRIP(ibuf,'T')
                     ELSE rotbuf = rotbuf STRIP(ibuf,'T')
    SELECT
      WHEN (LENGTH(rotbuf) = trunc.1) THEN DO /* perfect fit */
        'INPUT' rotbuf
        rotbuf = ''
      END
      WHEN (LENGTH(rotbuf) > trunc.1) THEN DO /* more than enough */
        DO FOREVER
          /* Find last blank, starting at TRUNC.1+1, working backwards */
          i = trunc.1+1
          DO WHILE (SUBSTR(rotbuf,i,1) ^= ' ')
            i = i - 1
            IF (i = 0) THEN SIGNAL word_too_long /* word > trunc */
          END
          'INPUT' SUBSTR(rotbuf,1,i-1)
          rotbuf = STRIP(SUBSTR(rotbuf,i),'B')
          IF (LENGTH(rotbuf) < trunc.1) THEN LEAVE /* Split enough? */
        END
      END
      OTHERWISE NOP /* not long enough - read another line */
    END       /* end of select */
  END
END
IF (rotbuf ^= ' ') THEN DO        /* Anything left in buffer? */
  'INPUT' rotbuf          /* put it out */
  rotbuf = ''
END
/* Clean up our toys and go home */
ADDRESS UNIX 'rm -f' tempfile
RETURN

/* Error routines */

word_too_long:
'INPUT ************************************************************'
'INPUT *ERROR* Justify encountered word longer than TRUNC setting.'
'INPUT Split word manually and restart justification from there.'
'INPUT Delete these error message lines.'
'INPUT ************************************************************'
CALL EXIT 13

EXIT:
PARSE ARG orc .
EXIT orc
```

MACROS FOR OTHER UTILITIES: INTEGRATE WITH UNIX OPERATIONS

PROVIDE BACKGROUND COMPILATION INITIATED FROM THE EDITOR, WITH THE RESULTING COMPILER ERROR MESSAGES DISPLAYED IN A POP-UP X WINDOW

```
/*
 * mk.xedit
 *
 * Runs make out of an xedit session.
 *
 * Default name is taken from source filename assumed to be of
 * the form "name.something".  So if you are editing key.c, this
 * routine will kick off "make key".  You can also issue "mk else"
 * if you want to make another target.
 *
 * The real work is done in a background task, and its output is
 * presented in a separate window.
 */
parse arg name
if name='' then do                  /* if name not specified, generate the default */
     "extract /fname"
     name=left(fname.1,pos(".",fname.1)-1)
     end
"save"                              /* make sure the disk file is up to date */
address unix "xemake" name "&"      /* kick off the background task */


#!/usr/local/bin/rxx
/*
 * xemake - run a make and display the results in a window.  Normally
 *          invoked from with xedit via mk.xedit.
 */
parse arg name                                  /* get name of make target */
"make" name ">"||name||".makeout 2>&1"          /* run make, output to a file */
"xt xe" getcwd()"/"||name||".makeout"           /* display the results */
```

General purpose programming in REXX

1. Reusable filters written in REXX vs. in-line AWK or SED programming

2. Small applications can be crafted by pulling together existing system facilities, integrated through REXX programming.

3. No high productivity language normally available in Unix. Alternatives are usually C and FORTRAN.

4. REXX applications can be ported to Unix from other platforms.

```
#!/usr/local/bin/rxx
/*
 * both - find lines containing both strings within a specific number
 *        of words.
 */
parse arg first second distance      /* two strings and a min. distance */
do while lines()>0
   line=linein()
   fpos=wordpos(first,line)             /* position of first word or 0 */
   spos=wordpos(second,line)           /* position of second word or 0 */
   if fpos>0 & spos>0 & abs(spos-fpos)<=distance
   then call lineout(,line)                      /* write matching lines */
   end
call lineout()                                    /* close output file */
exit


#!/usr/local/bin/rxx
/*
 * mult - find lines containing all input strings
 */
parse arg strings                          /* all words to be searched for */
do x=1 while lines()>0           /* x= only for leave instruction below */
   line=linein()                    /* line is a candidate to be tested */
   do i=1 to words(strings)                 /* try all words in string. */
      if wordpos(word(strings,i),line)=0          /* 0 means not found */
      then leave x                             /* terminates outer loop */
      end                                        /* end of all tests */
   call lineout(,line)                  /* if all found, write it out. */
   end
call lineout()                                    /* close output file */
exit
```

GENERAL PURPOSE PROGRAMMING: INTEGRATION OF EXISTING FACILITIES

THIS SAMPLE IMPLEMENTS A "PHONE DIRECTORY" BY USING XEDIT, DRIVEN BY A REXX PROGRAM. "PH <NAME>" POPS UP AN X WINDOW SHOWING AN EDIT SESSION THAT HAS BEEN PRE-POSITIONED ON THE FIRST LINE IN THE DATASET THAT CONTAINS <NAME>.

```
#!/usr/local/bin/rxx
parse arg name                          /* get the name he wants to find */
"rxx ph2" name "&"                     /* pass it along to the background */
```

```
#!/usr/local/bin/rxx
parse arg string                        /* get the name he wants to find */
"cp $HOME/.profile.xedit ph.xedit"          /* copy his .profile.xedit */
call lineout 'ph.xedit', '"cl/'string'"'  /* add a search command to it */
call lineout 'ph.xedit'                                /* close new profile */
"xt xe -p ph $HOME/phone/dir ] s"          /* xe phone/dir in a window */
"rm ph.xedit"            /* cleanup after synchronous window terminates */
```

General purpose programming: High productivity language

No high productivity alternative is usually present in Unix.
Alternatives are usually limited to C and FORTRAN.

```
#!/usr/local/bin/rxx
/*
 * vptrim - a utility to trim ventura publisher markup from a word
 *          processing file.
 *
 * vptrim infile traceopt
 *
 *        infile required, specifies the input file containing a
 *               word processing file that contains ventura publisher
 *               markup string.
 *
 *        traceopt optional, a trace instruction operand to turn on
 *                 REXX tracing.
 *
 * The output is sent to STDOUT, and may be redirected to a file.
 *
 * Example:  vptrim xehelp > xehelp2
 *
 * Most "@... = " and <...> sequences are simply removed from the
 * file.
 *
 * <T> is changed into three blanks.
 *
 * @FUNCTION = text is appended to the start of the next line, with
 * a " - " placed between the two chunks of text.
 *
 * << and >> are translated to < and > respectively.
 *
 * Room for improvement:
 *
 *   We could define our own set of tab stops and try to handle (T)
 *   in some smarter way.
 *
 *   @FUNCTION trick should maybe be extended to handle multiple such,
 *   through a table of special functions.
 *
 */
```

NO HIGH PRODUCTIVITY ALTERNATIVE IS USUALLY PRESENT IN UNIX.
ALTERNATIVES ARE USUALLY LIMITED TO C AND FORTRAN.

```
parse arg fn traceopt
trace value traceopt

if fn=""
then do
     say "usage: vptrim fn traceopt"
     exit
     end

lag=""

do while lines(fn)>0        /* push the entire file through this loop */

   line = linein(fn)

   do while pos("<T>",line)>0    /* turn <T> into white space */
      line=overlay("   ",line,pos("<T>",line))
      end

   do while pos("<<",line)>0     /* turn << into x'01' to hide them */
      line=left(line,pos("<<",line)-1)||'01'x||,
             right(line,length(line)-pos("<<",line)-1)
      end

   do while pos(">>",line)>0     /* turn >> into x'02' to hide them */
      line=left(line,pos(">>",line)-1)||'02'x||,
             right(line,length(line)-pos(">>",line)-1)
      end

   do while pos("<",line)>0      /* take out all other <..anything..> */
      if pos(">",line)>0
      then line=left(line,pos("<",line)-1)||,
             right(line,length(line)-pos(">",line))
      else do
           say '********* VPTRIM ERROR: Unmatched "<" in the following line.'
           leave
           end
      end

   line=translate(line,"<>","0102"x) /* unhide translated << and >> */

   if left(line,1)="@"               /* check for paragraph tag */
   then do
        type=left(line,10)           /* remember tag type */
        line = right(line,length(line)-pos("=",line)-1)  /* remove it */
        if type="@FUNCTION "
        then do                      /* for @FUNCTION tag */
             lag = line "-"          /* save text for next line */
             iterate                 /* and skip putting it out now */
             end
        end

   say lag line    /* put out current line plus any lag data */
   lag=""
   end
```

General purpose programming: Portability

REXX on Unix allows for porting applications developed on other platforms. Specific areas of concern:

- OS commands
- I/O facilities

For large programs, these areas can easily be a minor part of the code.

```
doit:
PARSE ARG targ
tempfile = 'JJ.TEMP'
ADDRESS UNIX 'rm -f' tempfile
'PUTD' targ tempfile
'UP 1'
'EXTRACT /TRUNC'

rotbuf = ''  /* initialize rotating buffer */
DO FOREVER
  IF (LINES(tempfile) = 0) THEN LEAVE    /* EOF? */
  ibuf = LINEIN(tempfile)
  IF (SUBSTR(ibuf,1,1) = ' ') THEN DO /* Paragraph break, either kind */
    IF (rotbuf ^= ' ') THEN DO    /* Anything left in buffer? */
      'INPUT' rotbuf    /* put it out */
      rotbuf = ''
    END
  END
  IF (ibuf = ' ') THEN 'INPUT   '    /* Blank line */
  ELSE DO                          /* duit tuit */
    /* concatenate the new stuff */
    IF (rotbuf = '') THEN rotbuf = STRIP(ibuf,'T')
                     ELSE rotbuf = rotbuf STRIP(ibuf,'T')
    SELECT
      WHEN (LENGTH(rotbuf) = trunc.1) THEN DO /* perfect fit */
        'INPUT' rotbuf
        rotbuf = ''
      END
      WHEN (LENGTH(rotbuf) > trunc.1) THEN DO /* more than enough */
        DO FOREVER
          /* Find last blank, starting at TRUNC.1+1, working backwards */
          i = trunc.1+1
          DO WHILE (SUBSTR(rotbuf,i,1) ^= ' ')
            i = i - 1
            IF (i = 0) THEN SIGNAL word_too_long /* word > trunc */
          END
          'INPUT' SUBSTR(rotbuf,1,i-1)
          rotbuf = STRIP(SUBSTR(rotbuf,i),'B')
          IF (LENGTH(rotbuf) < trunc.1) THEN LEAVE /* Split enough? */
        END
      END
      OTHERWISE NOP /* not long enough - read another line */
    END       /* end of select */
  END
END
IF (rotbuf ^= ' ') THEN DO        /* Anything left in buffer? */
  'INPUT' rotbuf          /* put it out */
  rotbuf = ''
END
/* Clean up our toys and go home */
ADDRESS UNIX 'rm -f' tempfile
RETURN

/* Error routines */

word_too_long:
'INPUT ************************************************************'
'INPUT *ERROR* Justify encountered word longer than TRUNC setting.'
'INPUT Split word manually and restart justification from there.'
'INPUT Delete these error message lines.'
'INPUT ************************************************************'
CALL EXIT 13

EXIT:
PARSE ARG orc .
EXIT orc
```

Embedded applications

1. Business applications
2. Utility software

Embedded applications

...require a robust API.

uni-REXX's API was modeled after that used by TSO/E REXX.

1. REXX program invocation from a C-based application
2. Ability to create addressable environments (i.e., SUBCOM)
3. Variable pool interface
4. C-based external functions (yet to be delivered)

Practical Application of REXX in the Unix Environment

- Supports migration of existing staff to Unix
- Brings new levels of integration and ease of use to Unix

Practical Application of REXX in the Unix Environment

2nd Annual SLAC REXX Symposium
Asilomar Conference Center
Pacific Grove, California

May 9, 1991

Presented by:
Ed Spire
The Workstation Group
Rosemont, Illinois

As commercial users migrate from proprietary IBM mainframes to Unix, they are often bringing REXX with them. REXX not only aids in the migration process, but also brings new functionality to the experienced Unix user. In many cases, you can use a single REXX program or macro where a native Unix solution would require a combination of tools (one of several shells, awk, grep, sed, etc.) each of which has its own syntax and ideosyncracies.

This presentation will discuss the applicability of REXX to various tasks in the Unix environment, and show examples where appropriate.

General types of applications:

1. Unix command macros

2. Macros for other utilities that use REXX, perhaps in combination with Unix facilities (primarily XEDIT)

3. General purpose programming in REXX

   custom filters

   entire applications

4. Embedded REXX applications

UNIX COMMAND MACROS are written to provide shorthand notations for often used, hard to remember, or lengthy sequences.

For example, sending something to a printer may require some local rain dance. Once you figure out what that particular rain dance is, you could "can" that research in a simple REXX program

```
#!/usr/local/bin/rxx
/*
 * rlp - print on a printer on another machine
 *
 * rlp filename machine traceopt
 *
 *      filename is the name of the file to be printed.
 *      machine is the machine that has the desired printer (defaults
 *              to scotty)
 *      traceopt is a rexx trace option, defaults to no tracing.
 */
parse arg fn machine traceopt
trace value traceopt
if machine="" then machine="scotty"
"cat" fn "| rsh" machine "lpr"
```

The command syntax for some Unix commands can be less than obvious. Once again, when you figure out a command syntax, you might want to "cover" the command with a REXX program that has a syntax that you find more intuitive.

Further, sometimes you find Unix commands that can be "misused". "find /" (i.e., find starting from the root directory) is often a really bad idea, since it will be looking through many relatively slow (i.e., remotely mounted NFS) file systems.

```
#!/usr/local/bin/rxx
/*
 * fi - run find on just /usr, where everything is anyway.
 *
 * this helps you not run find on the root, which would go out and
 * look through all your nfs mounts.  It also helps you not have to
 * remember the find command's syntax...
 */
parse arg name
"find /usr -name" name "-print"
```

The above are perhaps trivial to the experienced Unix user, but provide a handy shortcut for new Unix users. Once a method is researched, it is "canned" for future use. This also helps avoid one's shooting yourself in the foot as in find (root)...

Users who are migrating to Unix from proprietary IBM mainframe platforms are normally conversant enough with REXX to use it in the above manner as part of their migration efforts. Or, a thorough effort on the part of a support group could easily provide a series of such utilities that could ease a migration.

Some often used sequences are a bit more complex. In this case REXX is useful even for the experienced Unix user.

For example, when debugging an application which is using a terminal window in a fullscreen mode (perhaps via the curses terminal I/O package), it is nice to run the trace/debug package (dbx) from another window, so the debugging information won't interfere with the "fullscreen" I/O from the application. Normally, to do this, you display the list of active processes, scan it to find the process which represents the fullscreen application, and then invoke DBX supplying it with that process ID number. A fairly simple REXX program can automate that task for you.

```
#!/usr/local/bin/rxx
/*
 * dbxw - run dbx on the program running in another window.
 * This is useful when the program in the other window is a curses
 * application and the dbx output would mess up its "screen" display.
 *
 * dbx programname
 *
 * will run a ps -u userid and look for a process running programname,
 * and then dbx -a processid.
 *
 * Note that you should probably cd to the directory where the program
 * resides before you dbxw.
 */
parse arg programname traceopt
trace value traceopt
call popen "ps -u" userid() "| grep" programname
select
  when queued()=0
   then say "can't find" programname
  when queued()=1
   then do
        parse pull processid .
        "dbx -a" processid
        end
  when queued()>1
   then say "more than one" programname "running!"
  end
```

Some facilities are present but not easily used without special support. REXX is a convenient way to provide that support.

For example, the X windows system lets you control various aspects of the system by sending special character sequences to the terminal. It's pretty unlikely that someone is going to remember the special sequence that changes a window's title, for instance. But a REXX program could make it very easy to use this feature.

```
#!/usr/local/bin/rxx
/*
 * setnames - change the name associated with an X window.
 *
 *   The name of the window or its icon can be changed by sending
 *   a specific escape sequence to the terminal window...
 */
escape = x2c("1b")
parse arg name                                    /* charout avoids the */
call charout , escape||"]l"||name||escape  /* unwanted 'cr' that */
call charout , escape||"]L"||name||escape  /* lineout would send */
```

Typical usage of setnames:

```
#!/usr/local/bin/rxx
/*
 * rl - rlogin to another system, changing the names in the cterm window
 *      and icon to reflect that system's name
 */
parse arg system                                        /* who to rlogin to */
"setnames" system                                        /* put his name up */
"rlogin" system                                            /* rlogin to him */
call popen "hostname"                                        /* who are we? */
parse pull hostname
"setnames" hostname                          /* restore this system's name */
```

After a while you will have assembled a series of "local tools", building one upon the other, which can vastly improve the usability of Unix; and without climbing the rather steep learning curve associated with the various Unix utilities and filters which you would have to put together to accomplish all this without REXX.

Sometimes you might want to cover a Unix command that has many operands, with a syntax that makes the most often used operands more accessible. Now we're starting to write REXX programs with more than just one or two operands.

Before we proceed, let me mention how we write REXX macros that accept many operands, so that the user can invoke them with the same flexibility usually found in native commands.

Complex REXX programs at our installation use CPARSE to bring in operands...

```
/******************* PARSE SEQUENCE PATTERN ***************************/
/* MODIFY THE THIRD LINE AS YOUR "PROTOTYPE" SHOWING PARMS AND DFLTS  */
/******************* START OF PARSE SEQUENCE ***************************/
parse arg a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18
interpret cparse(,
"p1 p2(*) ( nk1 nk2 k1(k1v) k2() ",
||" )" a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18)
/******************* END OF PARSE SEQUENCE ***************************/
```

Cparse is an external REXX subroutine that accepts a string describing the model syntax of the REXX main program's parameter list, followed by the parameters that were actually passed to the main program. Cparse returns a string of assignment statements which, when interpreted by the main program, will place the appropriate symbols in the main program's symbol table to reflect the arguements that were passed to the main program (as parsed against the model syntax.)

Note that CPARSE was ported directly from CMS with no changes (other than case considerations), and hence implements a CMS-style command operand syntax. It could easily be modified to support a Unix-style syntax.

CPARSE makes it easy to write REXX programs that accept several operands, providing the flexibility found in native commands. This allows a REXX program to easily "cover" a basic Unix facility, reorganizing the parameter structure to the user's liking.

XTERM is one such command, with many operands, most of which you want to have standard values for, and a few which you might want to be able to change easily. XTERM has **lots** of operands...

```
xterm [ -ah] [ -ar] [ -b NumberPixels] [ -bd Color] [ -bg Color]
      [ -bw NumberPixels] [ -ccCharRange:Value[,...]]
      [ -cr Color] [ -cu] [ -display Name:Number] [ -dw]
      [ -fb Font] [ -fg Color] [ -fn Font] [ -fr Font]
      [ -fullcursor] [ -geometry Geometry] [ #Geometry] [ -help]
      [ -i] [ -ib File] [ -j] [ -keywords] [ -lang Language] [ -l]
      [ -leftscroll] [ -lf File] [ -ls] [ -mb] [ -mc Number]
      [ -ms Color] [ -n IconName] [ -name Application]
      [ -nb Number] [ -po Number] [ -ps] [ -reduced] [ -rv]
      [ -rw] [ -s] [ -sb] [ -sf] [ -si] [ -sk]
      [ -sl NumberLines] [ -sn] [ -st] [ -suppress] [ -T Title]
      [ -ti] [ -tm String] [ -tn TerminalName] [ -ut]
      [ -v] [ -vb] [ -W] [ -xrm String] [ -132] [ -e Command]
```

Examples

The following example can be used to create an xterm, specifying the size and location of the window, using a font other than the default, and also specifying the foreground color to be used of the text. It then runs a command in that window.

```
xterm -geometry 20x10+0+175 -fn Bld14.500 -fg DarkTurquiose
                            -e /tmp/banner_cmd &
```

Now if you want to establish local defaults for some of these and make the few operands you would normally use more accessible, you can cover XTERM with a REXX program like this...

```
#!/usr/local/bin/rxx
/*
 * xt - start an xterm window
 *
 * The positional parms comprise a Unix command that is to be run in this
 * window.  If none is specified, then your normal shell is run instead.
 *
 * optional parms:
 *   l #  - number of lines (default 25)
 *   x #  - x component of window location
 *   y #  - y component of window location
 *   i    - if present, window starts as an icon.
 *   fr # - reduced screen font size (default 14, the typical default
 *          for normal aixterm windows.)
 *   fn # - normal screen font size (default 10, small than typical for
 *          aixterm windows).
 *             The above two default settings make for normally small
 *             windows, which can be temporarily enlarged back to their
 *             traditional size by selecting "reduced" from the alt-
 *             left button menu.
 *   s    - if present, xterm is run synchronously.
 *   test - if present, the options line is shown on the screen and
 *          aixterm invocation is supressed.
 */
/******************** PARSE SEQUENCE PATTERN ****************************/
/* MODIFY THE THIRD LINE AS YOUR "PROTOTYPE" SHOWING PARMS AND DFLTS  */
/******************** START OF PARSE SEQUENCE ***************************/
parse arg a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18
interpret cparse(,
"c1() c2() c3() c4() c5() c6() ( l(25) x() y() i fr(14) fn(10) test s",
||" )" a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18)
/******************** END OF PARSE SEQUENCE *****************************/
cmd=c1 c2 c3 c4 c5 c6                          /* build the desired command */
if cmd="" then command = ""           /* build the required xterm option */
          else command = "-e" cmd
if s="" then amp="&"  /* build the required background execution option */
        else amp=""
options="-fullcursor -sb -sl 999 -ar -ls"  /* initial xterm options set */
options=options "-geometry 80x"l                /* number of lines option*/
if x<>"" | y<>"" then do                       /* if position specified, */
     if x="" then x=0                      /* fill in remaining defaults */
     if y="" then y=0
     options=options||"+"||x||"+"||y                 /* position option */
     end
options=options "-fn Rom"||fn||".500"              /* normal font option*/
options=options "-fb Rom"||fn||".500"                /* bold font option*/
options=options "-fr Rom"||fr||".500"             /* reduced font option */
if i="i" then options=options "-i"     /* if req, then start as an icon */
call chdir("/u/ets")                          /* back to home directory */
if test=""                                          /* show it or do it */
then "/usr/lpp/X11/bin/aixterm" options command amp
else say options command amp
```

Typical use of xt, others further below.

```
#!/usr/local/bin/rxx
/*
 * xi - initialize xterm environment
 *
 * this just creates my standard set of windows, with their normal
 * positions, but leaves them all as icons at first.
 */
"xt           ] x 0   y 0   i 1 64"              /* large primary window */
"xt           ] x 680 y 0   i"                   /* 1st alternate window */
"xt           ] x 680 y 395 i"                   /* 2nd alternate window */
"xt rl wrkgrp ] x 0   y 390 i"     /* window on wrkgrp (Sun-3) */
"xt rl drwho  ] x 575 y 630 i"     /* window on drwho (Sparc) */
"xt rl scotty ] x 600 y 590 i"     /* window on scotty (SCO/Unix) */
"xt rl orac   ] x 0   y 545 i"     /* window on orac (HP-9000/300 HP-UX */
"xt rl worf   ] x 0   y 545 i"     /* window on orac (HP-9000/300 Domain-OS*
/*
 * orac and worf are in the same spot, since they are the same machine,
 * and only one will be up at a time.  The other will die quitely
 * after a few attempts to rlogin
 */
```

Sometimes, you might want to extend the operating system's facilities. Here's an example associated with Network File System (NFS) usage. NFS can make remote machine file systems appear as local systems on your machine. You can see the relationships between local aliases for remote file systems in the df command output:

Filesystem	Total KB	free	%used	iused	%iused	Mounted on
/dev/hd4	49152	11356	76%	1146	9%	/
/dev/hd2	225280	37716	83%	6367	11%	/usr
/dev/hd3	32768	31700	3%	26	0%	/tmp
/dev/hd1	249856	223444	10%	1292	2%	/u
wrkgrp:/home	47946	5832	87%	-	-	/sun
wrkgrp:/usr	213313	18127	91%	-	-	/sunusr
wrkgrp:/	7608	2635	65%	-	-	/sunroot
drwho:/home	326519	66998	79%	-	-	/drwho
scotty:/	99037	4694	95%	-	-	/odt
drwho:/usr	183439	47344	74%	-	-	/whousr

So, in this example, if you are told that the file you need is available on machine drwho as /usr/local/bin, you would need to access the file as /whousr/local/bin. The aliases in use might vary from machine to machine.

We use the concept of a "system independent" filename, and have a routine that will compare the filename you specify to a 'df' command output, returning the local filename that will access the desired file from this machine. We then cover typical utility programs with a REXX invocation program that translates the filename through this routine, so that the user could refer to the above file as drwho:/usr/local/bin from any platform on the network.

```
/*
 * fn filename
 *
 * fn accepts a filename in a system independent form, and generates
 * a local filename which will provide (probably NFS) access to the
 * desired file.
 *
 * filename has the form
 *
 *   host:/filename/on/that.host
 *
 * note:  if host: is omitted, then no translation is done, assuming
 * that a local filename was really specified in the first place
 */
parse arg host ':' file
call popen 'hostname'
parse pull currenthost
select
when file=""                /* if no "host:", parse will have put it all */
then o=host                                 /* in host, and file will be null. */
when host=currenthost /* if explicitly referring to a file on this host */
then o=file                                          /* just use that file name */
otherwise do
 call popen 'df'
 lm=""                    /* will become the saved df line that matches. */
 do while queued()>0
  parse pull l
  parse var l dfhost ':' dfhostdir dfjunk '/' dflocaldir
  if length(dfhostdir)>0 ,                    /* weeds out header and locals */
      & dfhost==host                               /* weeds out other systems */
  then do
   if dfhostdir=="/"                                    /* for root file system */
   then lm=l                  /* save this line in case we find no other */
   else if left(file,length(dfhostdir))==dfhostdir       /* right line? */
   then do
    lm=l
    leave
    end
   end
  end
 if length(lm)>0                                       /* if we found something, */
 then do
  parse var lm dfhost ':' dfhostdir dfjunk '/' dflocaldir
  if dfhostdir^=='/'                                   /* if not root filesystem, */
  then file=right(file,length(file)-length(dfhostdir))  /* trim rmt dir */
  o='/'||dflocaldir||file                               /* add correct local dir */
  end
 else do                  /* if we found nothing, fail with error message */
      say 'sorry, no path from here to' host':'file
      return                          /* return with no value is a failure */
      end
 end
end
return o                                                /* non-failure return */
```

Typical usages of fn allow the user to access files on other systems without knowing the details of the NFS links that connect these systems.

```
#!/usr/local/bin/rxx
/*
 * nfl - invoke flist with system independent filename
 */
parse arg dir
ldir=fn(dir)
say 'flist' ldir
'flist' ldir
```

```
#!/usr/local/bin/rxx
/*
 * nxe - invoke xedit with a system independent filename
 */
'xe' fn(arg(1))
```

MACROS FOR OTHER UTILITIES THAT USE REXX can serve similar purposes.

We will use XEDIT as an example of a utility that uses REXX as its macro processor.

Once again, when you have figured out how to do something like print part of the edit file, you can "can" that procedure in a trivial REXX program.

```
/*
 * rlp.xedit - an xedit macro to print part of an xedit file
 *
 * rlp target
 *
 *   is the same as
 *
 *       !rm temp
 *       put target temp
 *       !rlp temp
 *
 */
parse arg target
address unix  'rm temp'
address xedit 'put' target 'temp'
address unix  'rlp temp'
```

A utility like XEDIT can also have significant functional extensions added to it via REXX programming. A typical requirement for a text editor is the ability to reform a modified paragraph to more nicely fit within it's margins. Here's a REXX macro that adds this function to XEDIT. Note that it was ported from CMS with minimal changes (to exactly three out of 66 lines of code.)

```
/* FLOW MACRO.
This macro aligns two or more lines of a text-type file
being edited (such as a NOTE). It tries to place as many
words as possible on a line, within the right margin
defined by XEDIT SET TRUNC.

USE:

    FLOW <target>

where <target> is a standard Xedit target defining the first line
not to be flowed. Typically, the alignment process will result in
there being fewer lines in the block than there were before alignment.
This will not always be true.

UNIQUE CAPABILITY OF THIS PROGRAM:

This macro can, unlike other parts of XEDIT, shorten lines. If you
SET TRUNC to a value shorter than some of the lines in your file,
they will be handled correctly by this macro. Elsewhere in XEDIT,
results are unpredictable and will likely involve data loss.

MODIFICATION HISTORY:

11/16/86 - Roger Deschner - Original version
01/02/88 - Roger Deschner - Replace call to "JOIN", for performance
02/14/88 - Roger Deschner - Allow lines to be shortened; use PUTD
10/24/89 - Roger Deschner - Protect from LINEND character
10/04/90 - Roger Deschner - Changed to FLOW; moved to RS/6000
*/

/* Do it to it */

doit:
PARSE ARG targ
tempfile = 'JJ.TEMP'
ADDRESS UNIX 'rm -f' tempfile
'PUTD' targ tempfile
'UP 1'
'EXTRACT /TRUNC'

rotbuf = ''  /* initialize rotating buffer */
DO FOREVER
  IF (LINES(tempfile) = 0) THEN LEAVE    /* EOF? */
  ibuf = LINEIN(tempfile)
  IF (SUBSTR(ibuf,1,1) = ' ') THEN DO /* Paragraph break, either kind */
    IF (rotbuf ^= ' ') THEN DO    /* Anything left in buffer? */
      'INPUT' rotbuf    /* put it out */
      rotbuf = ''
    END
  END
  IF (ibuf = ' ') THEN 'INPUT   '    /* Blank line */
  ELSE DO                          /* duit tuit */
    /* concatenate the new stuff */
    IF (rotbuf = '') THEN rotbuf = STRIP(ibuf,'T')
                     ELSE rotbuf = rotbuf STRIP(ibuf,'T')
    SELECT
```

```
      WHEN (LENGTH(rotbuf) = trunc.1) THEN DO /* perfect fit */
        'INPUT' rotbuf
        rotbuf = ''
      END
      WHEN (LENGTH(rotbuf) > trunc.1) THEN DO /* more than enough */
        DO FOREVER
          /* Find last blank, starting at TRUNC.1+1, working backwards */
          i = trunc.1+1
          DO WHILE (SUBSTR(rotbuf,i,1) ^= ' ')
            i = i - 1
            IF (i = 0) THEN SIGNAL word_too_long /* word > trunc */
          END
          'INPUT' SUBSTR(rotbuf,1,i-1)
          rotbuf = STRIP(SUBSTR(rotbuf,i),'B')
          IF (LENGTH(rotbuf) < trunc.1) THEN LEAVE /* Split enough? */
        END
      END
      OTHERWISE NOP /* not long enough - read another line */
    END       /* end of select */
  END
END
IF (rotbuf ^= ' ') THEN DO         /* Anything left in buffer? */
  'INPUT' rotbuf           /* put it out */
  rotbuf = ''
END
/* Clean up our toys and go home */
ADDRESS UNIX 'rm -f' tempfile
RETURN

/* Error routines */

word_too_long:
'INPUT ************************************************************'
'INPUT *ERROR* Justify encountered word longer than TRUNC setting.'
'INPUT Split word manually and restart justification from there.'
'INPUT Delete these error message lines.'
'INPUT ************************************************************'
CALL EXIT 13

EXIT:
PARSE ARG orc .
EXIT orc
```

The use of extensive REXX macros in a setting such as XEDIT becomes more viable on fast RISC processors, where the relatively low speed of its interpretive execution is not a problem. We expect to see REXX macros adapt XEDIT to many widely varied tasks (such as true word processing with automatic paragraph reform, LEXX-style live parsing, etc.) Such applications would be far too slow on the previous generation of computing platforms.

When using a utility that supports REXX, you also have the opportunity to integrate the work being done through the utility with work to be done in Unix itself.

When using XEDIT to write programs, the following REXX macro set will make it very easy to "kick off" a compilation, and put that compilation in the background, so that you can continue editing while the compiler checks your syntax. Once the compilation is completed, an X window is presented showing the output from the compiler.

```
/*
 * mk.xedit
 *
 * Runs make out of an xedit session.
 *
 * Default name is taken from source filename assumed to be of
 * the form "name.something". So if you are editing key.c, this
 * routine will kick off "make key". You can also issue "mk else"
 * if you want to make another target.
 *
 * The real work is done in a background task, and its output is
 * presented in a separate window.
 */
parse arg name
if name='' /* if name not specified, generate the default */
then do
     "extract /fname"
     name=left(fname.1,pos(".",fname.1)-1)
     end
"save" /* make sure the disk file is up to date */
address unix "xemake" name "&" /* kick off the background task */
```

```
#!/usr/local/bin/rxx
/*
 * xemake - run a make and display the results in a window. Normally
 *          invoked from with xedit via mk.xedit.
 */
parse arg name /* get name of make target */
"make" name ">"||name||".makeout 2>&1" /* run make, output to a file */
"xt xe" getcwd()"/"||name||".makeout" /* display the results */
```

GENERAL PURPOSE PROGRAMMING IN REXX fits quite nicely with the Unix philosophy of small, reusable pieces of code. Custom filters are easily written in REXX. These programs read STDIN and write part of the output file to STDOUT.

The Unix style of programming often uses data sources and filters in a multi-tasking "pipe" to achieve a particular result. Often custom filters are required, and in many cases they are created "on the spot" with utilities like sed, awk or perl. REXX can be used to write general purpose filters which can be documented and retained for future use.

Here are two examples, which filter a file, passing only lines that contain specific string combinations.

```
#!/usr/local/bin/rxx
/*
 * both - find lines containing both strings within a specific number
 *        of words.
 */
parse arg first second distance      /* two strings and a min. distance */
do while lines()>0
   line=linein()
   fpos=wordpos(first,line)             /* position of first word or 0 */
   spos=wordpos(second,line)           /* position of second word or 0 */
   if fpos>0 & spos>0 & abs(spos-fpos)<=distance
   then call lineout(,line)                        /* write matching lines */
   end
call lineout()                                      /* close output file */
exit


#!/usr/local/bin/rxx
/*
 * mult - find lines containing all input strings
 */
parse arg strings                          /* all words to be searched for */
do x=1 while lines()>0           /* x= only for leave instruction below */
   line=linein()                    /* line is a candidate to be tested */
   do i=1 to words(strings)                    /* try all words in string. */
      if wordpos(word(strings,i),line)=0              /* 0 means not found */
      then leave x                              /* terminates outer loop */
      end                                            /* end of all tests */
   call lineout(,line)                    /* if all found, write it out. */
   end
call lineout()                                      /* close output file */
exit
```

Often, small applications can be crafted by pulling together pieces of REXX code which bring existing system facilities together.

Here's a sample which implements a "phone directory" using XEDIT, driven by a REXX program. "ph name" pops up an X window showing an edit session that has been pre-positioned on the first line in the dataset that contains "name".

```
#!/usr/local/bin/rxx
parse arg name                             /* get the name he wants to find */
"rxx ph2" name "&"                        /* pass it along to the background */


#!/usr/local/bin/rxx
parse arg string                          /* get the name he wants to find */
"cp $HOME/.profile.xedit ph.xedit"              /* copy his .profile.xedit */
call lineout 'ph.xedit', '"cl/'string'"'  /* add a search command to it */
call lineout 'ph.xedit'                                /* close new profile */
"xt xe -p ph $HOME/phone/dir ] s"            /* xe phone/dir in a window */
"rm ph.xedit"            /* cleanup after synchronous window terminates */
```

And, as always, low volume applications can be developed and maintained much more cost effectively with REXX than with many other languages. This is especially true in the Unix world where the only universally available language is C, and the most likely alternate is FORTRAN, and BASIC is not usually present.

```
#!/usr/local/bin/rxx
/*
 * vptrim - a utility to trim ventura publisher markup from a word
 *          processing file.
 *
 * vptrim infile traceopt
 *
 *        infile required, specifies the input file containing a
 *               word processing file that contains ventura publisher
 *               markup string.
 *
 *        traceopt optional, a trace instruction operand to turn on
 *                 REXX tracing.
 *
 * The output is sent to STDOUT, and may be redirected to a file.
 *
 * Example:  vptrim xehelp > xehelp2
 *
 * Most "@... = " and <...> sequences are simply removed from the
 * file.
 *
 * <T> is changed into three blanks.
 *
 * @FUNCTION = text is appended to the start of the next line, with
 * a " - " placed between the two chunks of text.
 *
 * << and >> are translated to < and > respectively.
 *
 * Room for improvement:
 *
 *   We could define our own set of tab stops and try to handle (T)
 *   in some smarter way.
 *
 *   @FUNCTION trick should maybe be extended to handle multiple such,
 *   through a table of special functions.
 *
 */

parse arg fn traceopt
trace value traceopt

if fn=""
then do
     say "usage: vptrim fn traceopt"
     exit
     end

lag=""

do while lines(fn)>0       /* push the entire file through this loop */

   line = linein(fn)

   do while pos("<T>",line)>0   /* turn <T> into white space */
      line=overlay("   ",line,pos("<T>",line))
      end

   do while pos("<<",line)>0    /* turn << into x'01' to hide them */
```

```
   line=left(line,pos("<<",line)-1)||'01'x||,
          right(line,length(line)-pos("<<",line)-1)
   end

do while pos(">>",line)>0     /* turn >> into x'02' to hide them */
   line=left(line,pos(">>",line)-1)||'02'x||,
          right(line,length(line)-pos(">>",line)-1)
   end

do while pos("<",line)>0      /* take out all other <..anything..> */
   if pos(">",line)>0
   then line=left(line,pos("<",line)-1)||,
          right(line,length(line)-pos(">",line))
   else do
        say '********* VPTRIM ERROR: Unmatched "<" in the following line.
        leave
        end
   end

line=translate(line,"<>","0102"x) /* unhide translated << and >> */

if left(line,1)="@"               /* check for paragraph tag */
then do
     type=left(line,10)           /* remember tag type */
     line = right(line,length(line)-pos("=",line)-1)  /* remove it */
     if type="@FUNCTION "
     then do                      /* for @FUNCTION tag */
          lag = line "-"          /* save text for next line */
          iterate                 /* and skip putting it out now */
          end
     end

say lag line    /* put out current line plus any lag data */
lag=""
end
```

REXX Applications that were developed on the mainframe can be ported to Unix without a complete rewrite in another language. You still have to pay close attention to the system interfaces, such as OS commands and I/O facilities.

The largest such application we have seen ported to Unix is a "silicon compiler" used to design microchips - a **very** large REXX program that was ported with minimal difficulties to Unix, and would have required a major rewrite in C or Fortran had REXX not been available.

APPLICATIONS THAT EMBED REXX are being ported to (or developed for) Unix. We have seen a refinery simulation system that embedded REXX as the simulator's control language ported to Unix with minimal effort. More than one vendor is working on automated operations and/or network control systems which embed REXX as the controlling language. REXX has been specified as the controlling language for other commercial Unix applications as well.

These applications require a robust API, similar to that found in VM/CMS or MVS/TSO. The uni-REXX API includes REXX program invocation, the ability to create an addressable environment (i.e., SUBCOM), and the Variable Pool Interface. Still to come are external functions written in C.

SUMMARY:

The availability of REXX in the Unix environment not only supports the migration of existing staff from proprietary mainframes to Unix, but also brings a new level of integration and ease of use to the Unix environment.