Linux Shell Scripting for the z/VM Rexx User

- Share 106, Seattle Washington March 2006
- Session 8320
- Gordon Wolfe, Ph.D.
  vmlinux@locolobo.fastmail.fm

Disclaimer

- Use the contents of this presentation at your own risk. The author does not accept any liability, implicitly or explicitly, direct or consequentially, for use, abuse, misuse, lack of use, misunderstandings, mistakes, omissions, mis-information for anything in or not in, related to or not related to, pertaining or not pertaining to this document, or anything else that a lawyer can think of or not think of.
- This material is based on the personal experiences of the Author and does not constitute a recommendation for any product or service by the author or any other entity.

Scope

- VM User new to running Linux? “I know how to write REXX execs. How do I write Shell scripts?”
- Unix user new to VM? “I know how to write Shell Scripts. How do I write REXX execs?”
- This is not a tutorial on basic CM/CMS commands or Linux shell commands. I will presume you know those already.
- This will be very simplistic. Lots more finer details. Buy a book.
- No guarantees I haven't fat-fingered some examples.

Rexx vs.Shell

- Rexx
  - There is only one rexx for VM. Similar REXX's on other platforms.
  - EXEC is early (pre VM/SP 4) version, not easily programmable
  - EXEC2 like MVS PLIST
- Shell
  - Many shells for *nix:
  - sh, bsh, bash, ksh,csh,tcsh, zsh
  - Linux usually uses bash
  - All similar, but different
  - REXX available for linux in Regina if you're lazy and don't want to learn shell scripts.

Start it Up!

- REXX
  - First Line must be a comment: /*…..*/
  - Recommend use of ‘address command’ - forces CP commands and called execs to be prefaced by ‘CP’ and ‘EXEC’
  - Help command
- Shell
  - First line must be #! /bin/bash
  - Tells what environment to run shell in
  - man command

Naming Conventions

- REXX filetype must be ‘EXEC’
- Execute by entering filename or “EXEC” filename
- Must exist on accessed disk in search order. Will take first one it finds. MYJOB EXEC A will run instead of MYJOB EXEC S. Be careful!
- Shell: Any filename
- Must be executable rwx rwx rwx
- Must exist in PATH
  - echo $PATH
- Or explicitly declare path “/usr/bin/command” or ./command on current directory

Capturing Output

- Rexx:
- CP SPOOL CONSOLE START TO *
- EXEC MYREXX
- CP SPOOL CONSOLE STOP CLOSE
- Results in virtual reader

- Script:
- script
- ./myshell
- exit
- Results in file “typescript”

Tracing

- Second line or later:
- trace r
- trace i for intermediate detail
- trace o to turn off

- First line:
- #! /bin/bash -x

Variables

- Any non-rexx command
- Use a, artifice, glop
  - Not if, do, end
  - Upper, lower case
- Declare by
  - Glop = ‘thing’
  - Glop = 2
- Generally not usable outside of exec (See GLOBALV)
- Preceded by $
- Convention: Uppercase
- Declare without $
  - GLOP=‘thing’
  - Echo $GLOP
- To use outside of script
  - Export $GLOP

Passing Arguments

- EXEC GLOP a b c
- parse arg a b c .

- ./glop a b c
- $1 contains a
- $2 contains b
- $3 contains c

Read in and Display User data

- say ‘What is your name?’
- parse pull name
- say name

- echo -n “What is your name?”
- read NAME
- echo $NAME

Variable Substitution

- glop = 'some text'
- glop = 4+3
- glop = oldglop||'more text'
- glop = oldglop*7
- Integer or FP arithmetic

- GLOP=Peach
- GLOP="Peach"
- GLOP=$((5+3*2))
- Integer arithmetic only. Results truncated to lower integer.

Standard In/Standard Out

- Rexx assumes
  - input from terminal (or stack)
  - Output to terminal
  - errors to terminal
- Can be redirected with
  - Pipes
  - EXECIO
  - FILEDEFs
  - CP SPOOL

- Stdin is terminal (0)
- Stdout is terminal (1)
- Stderr is terminal (2)
- Can be redirected!
  - Command 1>file1 2>file2

Getting Command Output into a Variable

- Several ways:
  - Use the Stack
    - ID (FIFO
    - parse pull user . node .
  - EXECIO (also stack)
  - PIPELINES
    - (More later)
- Use backquotes:
  - DATE=`date`
  - echo $DATE

If-Then

- if x='large' then
  - do
    - say 'x is big'
  - end

- if test "$X" = "large" ; then
  - echo "X is big"
- fi
- Or test string1=string2
- Or [string1 = string2]

If-Then-Else

- if x='large' then
  - do
    - say 'x is big'
  - end
- else do
  - say 'x not so big'
  - R = 8
- end

- if test "$X" = "large" ; then
  - echo "X is big"
- else
  - echo "X not so big"
- Fi

Types of Equality

- Strings
  - if x='large'
  - if x <> 'large'
  - if x='' (zero length)
- Numbers
  - if x=y
  - if x > y
  - if x >= y
  - if x <> y

- Strings
  - test str1 = str2
  - test str1 != str2
  - test -z str
- Numbers
  - [int1 -eq int2]
  - [int2 -gt int2]
  - [int1 -ge int2]
  - [int1 -ne int2]

Compound Equalities

- If x=y & w=z then …
- If x=y | w=z then ...

- If ["$X" = "$Y" && "$W" = "$Z"] ; then
- If ["$X" = "$Y" || "$W" = "$Z"] ; then

Select from List

- select
  - when x=1 then …
  - when x=2 then
  - otherwise …
- end

- case "$X" in
  - 1) do … ;;
  - 2) do … ;;
- esac
- If no matches found, does nothing.

Loops

- do x=1 to 20
  - …
- end
  - -or-
- x=1
- do until x=20
  - …
  - x = x+1
- end

- X=1
- while [$x -lt 20]
- do
  - …
  - X=`expr $X + 1`
- Done

Reading a file one line at a time

- do forever
  - 'MAKEBUF'
  - buf1 = rc
  - 'EXECIO 1 DISKR' fn ft fm
  - if rc <> 0 then leave
  - parse pull line
  - <manipulate line>
  - 'DROPBUF' buf1
- End
- 'DROPBUF' buf1
- 'FINIS' fn ft fm

- while read LINE
  - do
  - <manipulate LINE>
  - done < filename

Arrays

- var.1 = 1
- var.2 = 2
- var.3 = 5
- var.apple='peach'
- var.0 contains the number of items in the array
- say fruit.2 gives orange

- Name[index]=value
- Index must be integer => 0
- Arrays start index=0
- Index of * means all members
- Array variable accessed as:
  - ${name[index]}
  - Echo ${fruit[2]} gives orange
  - Echo ${fruit[*]} gives apple banana orange

Pipelines

- VERY complex and powerful
- Uses "stage" commands written just for pipes
- Can have multipath pipelines and pipes that call other pipes.
- PIPE stage | stage | stage
- PIPE literal 'hello' | console
  - Gives 'hello'
- Place all but rexx variables inside single quotes
- Useful:
  - 'PIPE CP QUERY NAMES | split at , | strip | locate /LNX/ | console
- Uses standard linux commands
- Directs stdout of one command to stdin of next command
- ps -ax | grep dsm
  - shows all running processes containing string "dsm"
- Useful:
  - tar -cf . | tar -xpf -C /mnt
  - Copies all files and subdirectories recursively from local directory to /mnt, preserving ownerships, dates and permissions. Quite fast.

Output Redirection to or from a file

- Use EXECIO, stack
- PIPEs is better:
- 'PIPE CP QUERY NAMES | split at , | strip | >LINUX SERVERS A'
- Use >> to append to existing file
- 'PIPE < LINUX SERVERS A | console'
- 'PIPE < LINUX SERVERS A | CP SIGNAL SHUTDOWN'
- ps -ax | grep dsm > dsm.processes
- Use >> to append to existing file
- Redundant, but illustrates the point: use a file as input to a command:
  - cat 0< .profile
  - mail friend@berkeley.edu < exam.answers

Quoting

- Rexx assumes variable if not quoted, literal if quoted.
- If variable not set, variable = name of variable
- say glop
  - glop
- glop = 'pudding'
- say glop
  - pudding
- say "glop"
  - glop
- Use single or double quote, but be consistent. Close with same type you open with.

- Quotes are special characters modifying what follows, like \ " '
- This is a very complex topic. Look it up.
- "meta-caracters" need to be quoted if not to be used as meta. Sometimes called "escaping" the character.
  - * ? [ ] ' " \ $; & ( ) | !
- Double quotes disable meta characters
  - Echo '$USER owes <-$1250,**>; [as of (`DATE %m %d`)]"
  - Rewrite as (use \ to escape $)
  - Echo '$USER owes <-\$1250,**>; [as of (`DATE %m %d`)]"
  - Gives
  - Fred owes <1250.**>; as of (1221)}

Return Codes - A way of checking success

- Special variable rc
- rc = 0 if ok
- rc <> 0 if not ok

- Special variable $?
- $? = 0 if ok
- $? != 0 if not ok

Subroutines

- 'ERASE PROFILE ANY A'
- call sub1 rc
- say result
- …
- sub1: procedure /*variables not visible outside subroutine */
- parse arg v1
- if v1 = 0 then return 'OK'
- else return 'bad juju'

- insmod -f "/lib/modules/misc/cmsfs.o" > /dev/null
- failed $? 1
- …
- failed() {
  - if [$1 -ne 0]; then
    - echo "failed to insmod cmsfs"
    - exit 0
  - fi
- }

Does a File Exist?

- 'ESTATE' fn ft fm
  - or
- 'ESTATEW' fn ft fm if you want to know if it's writeable
- rc = 0 if exists, writable
- rc = 28 otherwise
- fm must be in accessed disk list or else rc = 36

- if [-f /home/mystuff]; then
  - echo "exists"
- Will search for file whether in $PATH or not.
- Other options:
  - -e file or directory exists
  - -s exists and size > o
  - -w exists and writable
  - -x exists and executable

There's More…

- Both languages have MANY other options. This is just to get you started.
- Didn't cover:
  - Print formatting
  - Setting defaults for variables and arguments
  - CSL or Script libraries
  - Manipulation of strings
  - Interfaces to DB2, Oracle, other databases
  - Sleep, signals, wakeup

Further Reading:

- Shell:
  - man shell
  - Srirang Veeraraghaven "Sams Teach yourself Shell Programming in 24 Hours" SAMS publications (2002)
  - Stephen Kochan & Patrick Wood "Unix Shell Programming" Hayden Books (2004)
  - http://www.injunea.demon.co.uk/pages/page201.htm
- REXX
  - M.F.Cowlishaw, "The REXX Language, A Practial Approach to Programming" Prentice-Hall (1990)
  - SC24-5465-02 REXX/VM User's Guide (IBM)
  - SC24-5770-01 REXX/VM Reference (IBM)
  - SC24-5970-00 CMS Pipelines User's Guide (IBM)