THE BOURNE SHELL

Ken Wong
Washington University, St. Louis

kenw@arl.wustl.edu
www.arl.wustl.edu/~kenw
Copyright 2002

SHELL

• A shell is a command line interpreter
  * Primary Purpose: Translate command lines typed at a terminal
  * Input to a shell is from the terminal (interactive shell) or a file
  * The shell is determined by a user’s entry in the /etc/passwd file

• Some Shells
  * Bourne Shell (sh)
    • What I use mostly for writing shell scripts
  * C-Shell (csh)
    • Typically, the default command line interpreter
  * TC-Shell (tcs)
    • C-shell with file name completion and command line editing
  * Bash Shell (bsh)

• Shell Script: A file containing shell commands

WHY BOURNE SHELL?

• www.mars.org/home/rob/docs/csh-whynot.html

• sh(1) (i.e., ‘man sh’)

• Csh Difficulties
  * File descriptor manipulation
  * Lack of command language orthogonality
  * Can only trap SIGHRT

• Bourne Shell
  * Ubiquitous
  * Used in system-level scripts (e.g., /etc/rc.local)
  * But lacks many interactive features found in other shells

OVERVIEW OF sh

• The shell program sh is a macro processor that provides:
  * Parameter Substitution: $Name (e.g., $x)
  * Command Substitution: ‘Command...’ (e.g., ‘pwd’)
  * File Name Generation: * ? [. . .] (e.g., $F=”*”)

• Before command execution, the following substitutions occur:
  * Parameter Substitution; Command Substitution; Field Separator Interpretation
    • Field separators are defined by $IFS and by default include the space, tab
    and newline characters
  * File Name Generation

• A newline in a list acts like ‘;’

• Characters with special meaning: ; & ( ) | < > newline space tab
EXAMPLE 1 (SIMPLE FILE BACKUP)

- **Features:** Copy a list of files to the directory OLD/

- **Example**
  
  ```
  mkdir OLD;  # make the OLD directory
  cp x.c OLD;  # copy the 2 files x.c, y.c
  cp y.c OLD;
  ```

- **Alternatives**
  
  ```
  mkdir OLD; cp x.c y.c OLD;  # use multilple syntax of cp
  mkdir OLD; cp [xy].c OLD;  # uses file name generation
  ```

- **Shell Script**
  
  ```
  # Create file (e.g., bkup1)
  #!/bin/sh
  cp x.c y.c OLD
  ```

RUNNING THE SCRIPT

- **As an argument to a shell command**
  
  ```
  % sh bkup1 x.c y.c
  % sh bkup1 [xy].c
  ```

- **Make the file executable**
  
  ```
  % chmod 755 bkup1  # or 'chmod ugo+x bkup1'
  % chmod a+x bkup1'
  ```

- **Use the script**
  
  ```
  % bkup1 [xy].c  # or 'bkup1 x.c y.c'
  ```

PATTERNS (FILE NAME GENERATION)

- `*`: Matches any string including the null string
- `?`: Matches any single character
- `[..]`: Matches any one of the enclosed characters
- `[a-z]`: Matches any one of the characters between 'a' and 'z'

INPUT OUTPUT

- `< File`: Use File as standard input (file descriptor 0)
- `> File`: Use File as standard output (file descriptor 1)

- `<< Word`: Read from the current file until end of file or encountering a line containing Word alone
- `>> File`: Append standard output to File

- `Digit1>&Digit2`: Make file descriptor Digit1 a duplicate of file descriptor Digit2 (e.g., `wc foo.c > wc.out 2>&1`)
A BETTER SCRIPT

- **Purpose:** Copy files to ./OLD with a suffix of 0, ... , 9 appended to the name.

- **Usage:** bkup File1 File2 ...

- **Example:** bkup x.c y.c z.c
  
  * Copy files x.c, y.c, and z.c to ./OLD directory.
  * Append the single-digit suffix to the base filename, smallest digit first

- **Method (Pseudo-Code):**
  
  for each file F listed on the command line {
    for each suffix S from 1 to 9 {
      if file $F.$S does not exist {
        Copy file $F$ to ./OLD/$F.$S;
        break;
      }
    }
  }

A BACKUP SCRIPT

```
01 #!/bin/sh
02 # Usage: bkup2 File1 File2 ...
03 FILES=$*;           # all command-line args $1, $2, ...
04 for F in $FILES
05 do
06   for S in 0 1 2 3 4 5 6 7 8 9
07   do
08     D=OLD/$F.$S;
09     if test ! -f $D; then  # not a regular file
10       echo "Copying $F to $D";
11       cp $F $D;
12       break;
13     fi
14   done
15 done
```

PARAMETER/COMMAND SUBSTITUTION

- **Example 1 (Parameter Substitution)**

  F=foo;
  
  # F is a variable
  cp $F.3 $F.4;
  cp $F.2 $F.3;
  cp $F.1 $F.2;
  cp $F.0 $F.1;

- **Example 2 (Command Substitution)**

  WL="";
  
  # null string
  FL="ls *.txt";
  
  # evaluate command and assign
  for F in $FL
    do W="wc $F";
    WL=$WL " $W;
  done

AFTER SUBSTITUTION

- **Example 1**

  F=foo;
  
  cp foo.3 foo.4;
  cp foo.2 foo.3;
  cp foo.1 foo.2;
  cp foo.0 foo.1;

- **Example 2**

  WL="";
  
  FL="foo.txt goo.txt";
  
  # if only foo.txt and goo.txt in dir
  for F in foo.txt goo.txt
    do W="wc $F";
    WL=$WL "$W;
  done
PARAMETER SUBSTITUTION

- \${Parameter} indicates parameter substitution
  * The result of substitution is the value of the parameter
  * e.g., \$x

\[
\begin{align*}
  x &= \text{foo}\quad &\text{# Output is null string} \\
  \text{echo } \$xX &\quad &\text{# Output is } \text{foo}X
\end{align*}
\]

- Parameters
  * A Name: A sequence of letters, digits or underscores starting with a letter
  * A Number: \$1 is the first command line argument, \$2 is the second, etc.
  * Special Parameters: \@ \# \? \- \$

ANOTHER BETTER SHELL SCRIPT

- **Purpose:** Copy files to \./OLD with today's date and a version suffix appended to the name.
  * `bkup foo` might create the file `./OLD/\text{foo.09.10.02b}`

- **Usage:** `bkup [-s] File1 File2 ...`
  * `-s` flag: Do the work silently

- **Method:**
  Process command line flags and Exit if error;
  Create \./OLD directory if it doesn't exist;
  for each file F listed on the command line {
    if file \$F exists {
      \$S = Unused suffix;
      Copy F to \./OLD/\$F.$D\$S where \$D is the date;
      break;
    }
  }

ANOTHER BOURNE SCRIPT (Part 1)

```
01 #!/bin/sh
02 VERB=$1;
03 ERROR=;
04 while test "$1" != "" ; # process flags
05 do
06   case \$1 in
07     -s) VERB="" ; \; ;
08     -*) echo "Bad flag: \$1 ; ERROR= ; \; ;
09       *) break \; ;
10     esac;
11    shift;
12    done;
13    if test \$ERROR= ;
14    then
15      echo "Quitting because of errors";
16      exit 1;
17    fi;
18     FILES=*
19     SUFFIXES="a b c d e f g h i j k l m n o p q r s t u v w x y";
20     ENV="z";
21     SUFFIXES="$SUFFIXES" "$END";
22     DATE="date '+%m.%d.%y'";
23     if test ! -d OLD ; then \# create \./OLD if it doesn't exist
24      \# create \./OLD/\text{DATE} needs to exist"
25      mkdir OLD; chmod 755 OLD;
26      fi;
```

ANOTHER BOURNE SCRIPT (Part 2)

```bash
27 for f in $FILES  # for each file
28 do
29   B="/old/$f.DATE";  # new name without suffix
30   if test -f $f;  # must be a regular file
31   then
32     for S in $SUFFIXES  # find unused suffix
33     do
34       if test -f $f$S;  # suffixes a-y all used
35         then
36           echo "I give up on $f: Already 27 backups today.";
37           break;
38         fi;
39       if test -f $f$S; then  # suffix already used
40         continue;
41       else  # found unused suffix
42         if test $f$B$S; then
43           echo "Copying $f to $f$S";  # make copy
44         fi;
45         cp $f $f$S;
46         break;
47       fi;
48     done;
49   else
50     echo "The file $f does not exist.";
51     fi;
52 done;
```

QUOTING

- **Quoting** is indicated by ", '"', or "...
- **Strict Quoting**
  - Single Character: Backslash (\)
  - Multiple Characters: Single Quotes ('...',)
- **Loose Quoting**
  - Double Quotes ("")
  - Use backslash (\) to quote ", " $".
- What if $F is the null string???
  - if test -z $F then ...
- The Solution
  - if test -z "$" then ...

SYNTAX SUMMARY

```
CommandList ::= AndOr
AndOr ::= CommandList ;
CommandList & Command
AndOr & AndOr
CommandList & AndOr
PipeLine ::= Command
PipeLine | Command
Command ::= SimpleCommand ( CommandList )
              ( CommandList )
              for Name do CommandList done
              for Name in Word ... do CommandList done
              while CommandList do CommandList done
              until CommandList do CommandList done
              case Word in CasePart ... esac
              if CommandList then CommandList ElsePart fi
SimpleCommand ::= Item
                  SimpleCommand Item
```

```
Item ::= Word
InputOutput
Name = Value

Word ::= A sequence of non-blank characters
InputOutp ::= > Filename
           < Filename
           >> Word
           <= Word

Name ::= a sequence of letters, digits or underscores starting
       with a letter
```