Shell Programming

* Shell: a command interpreter.

loop:
ourput a prompt %;
accept a command;
if the command execution file exists (search)
then
begin
execute it;
(shell is waiting; fork another process to execute it)
return the status;
end
else an output error message;

When search a command execution file, shell will follow the direction specified in PATHs.

* UNIX (multiusers; multitasks)
hardware --> kernal (resource manager; scheduler) --> shells

* Command Hierarchy in UNIX

a) system calls. (the lowest level)
used in the kernal; its internal implementation will handle the details of the hardware.
EX. read; write.

b) C library.
EX. in <stdio.h>, fprint, fopen.

c) UNIX command.
EX. who, cat, ls.

d) shell command file. (a sequence of commands)

* Bourne shell (sh; the Bell Lab. version);
  C shell (csh; the UC Berkeley BSD version).

* Four ways to solve a problem in UNIX:

a) a single command. (+ options)
$ ls -i -l good bad \<return>
> ok <return>
...

b) command combination.

- I/O redirection.
$ cat file1 file2 > file3
$ cat file1 file2 >> file3
$ program1 <data1 >out1

- Pipe.
$ grep words file1 |sort > out
This is equal to:
$ grep words file1 >temp1
$ sort temp1 >out
$ rm temp1

(* cat, sort, grep are filters which can be used in pipe;
a filter can read data from standard input and write results
to standard output *)

c) a shell command file.
a shell program ex1 is as follows:

```
echo    Here is    the date and time:
date
```

```
$sh ex1    (* execute a shell program *)
Here is the date and time:
Mon Aug 16 8:20:34    1993
$
```

```
$ chmod u+x ex1 (* the other way to execute it *)
$ ex1
...
```

good: portable; occupy small space.
(make use of well-defined commands)
bad: slow execution speed as compared to a C program.

d) a C program.
can call C library, system calls.

* What can shell support us ?

1) pattern matching.
2) variables.
3) control structure (for, while, until, if, case).

* Pattern

1) ?
EX. d?g ==> dog, dig.

2) *
EX. s*n ==> sn, soon, sun.

3) [ ]
EX. [a-mACK0-9] ==> acK, mabAK43.

* Shell Variables

name=value (* no space in both sides of = *)
(* value is treated as a string *)
weight=33
view='' (* view= or view="" ; empty string *)
program = pp !! Error !!
big-foot=89 !! Error !!
big=small-foot !! OK !!
$ echo weight
weight
$ echo &weight
33 (* &name returns the value of the variable name *)
& echo My weight is &weight
My weight is 33

* System variables in shell

$cat .profile
Home= /usr/changyi
TERM=adm5
PATH=.:/bin: /bin: /usr/bin
PS1=$

* Interactive shell programs

    Type the following shell program:

    ```
    echo Hi, what is your name ?
    read name
    echo Hi, $name, nice to meet you !
    ```

* Non-interactive shell programs --parameters

    $#: the number of parameters.

    $0: the shell program name.

    $1, $2, ..: the first parameter, the second ...

    $*: all the parameters ($1, ..).

    $?: command execution status. (0 --> OK)

    EX. cp $1 $1.old

* set + parameters

    ```
    $ set pp tt ww
    $ echo $1 $2
    pp tt
    ```

* `command` (* backward quote *)

    It means to execute the commad.

    ```
    $ echo date
    date
    $ echo `date`
    Mon Aug 16 8:20:34    1993
    $ set `date`
    $ echo $1 $2
    Mon Aug
    ```

* More about commands

a) $ ls -l; cat file1 file2 >file3; date
(* a sequence of commands *)

b) $ cc ex.c&
(* a background command; no space before & *)

c) $ (date; cat results) >file1
(* grouping commands *)

d) $ cp file1 file2 2> errorfile
(* standard error file 2> ; 0> standard input; 1> output *)
$ sort fil1 >file.out 2> errorfile&

e) $ command1 && command2
(* if command1 is ok then command2 *)
$ command 1 || command2
(* if command1 is not ok then command2 *)

f) {variable-word}; {variable=word};
{variable+word}; {variable?message}

g) $ echo * \? \\ \[
* ? \ [
$ echo '*?'
*?
(* single quote removes the special meaning of the symbols for patten matching; it considers the content is a string *)
(* double quote remove the special meaning of some of symbols for patten matching, except $ ` ' \; it also evaluate variables $name and execute the command `command`*)
$ myname=peter
$ echo `myname`
peter
$ echo ' `myname` is big & smart'
`myname` is big & smart
$echo " `myname` is big & smart"
peter is big & smart

h) a shell file quotes is as follows:
cat "$*"
cat "$@"
$ quotes pp tt
==> cat "pp tt"; (cat "pp"; cat "tt")

i) # this is a remark

* Control Structure

* for

a) for name in aa bb cc do ... done

b) for i in $*
do
echo $i
done
(* for i *)
EX.

```
# send the same mail to several persons
# mails file name1 name2 ...
file=$1
shift
for name in
do
    mail $name <$file
done
```

c) for file in * do cat $file done

d) EX.

```
# files is a sequence of names
echo Hi, input files
read files
for n in $files
do
   echo $n
done
```

e) for i in `cat file1` do ... done

* EX.

```
for file in $*
do
   spell $file > sp.$file
done
```

* EX. a shell program ex1

```
# greps words_file file1 file2 ...
source=$1
shift
for i in `cat $source`
do
   grep $i $*
done
```

```
$sh ex1 filedate ch.[1-9]
```

* case

```
echo Hi, input name
read name
case $name in
a|b|c?) command1s ;;
[0-9][0-9]?[b-d]*k) command2s ;;
*) command3s ;;    (* otherwise *)
esac

case $1 in ...

for file in *
do
   case $file in
   *.old) rm $file ;;
   *.c) mv $file new.$file
   *)    ;;
   esac
done
```

* while

```
while control_command (* while execution is OK *)
do
   commands
done
(* check the return status of the command executed *)
(* different from Pascal while *)
```

* until

```
until control_command (* do until not ok *)
do
   commands
done
```

* if

```
if control_command (* if the status is OK *)
then
   commands
fi

if control_command
then
   commands
else
   commands
fi

if control_command
then
   commands
elif control_command2
then
   commands
else
   commands
fi
```

* EX.

```
if cmp -s $1 $2
then
   rm $2
fi
```

* Control Command TEST

a) test string1 = string2
(* there is a space on both sides of = *)
EX. if test $1 = peter
if test $name != peter
if test -n string (* if the length of the string≠0 *)
if test -z string (* if the length of the string = 0 *)
if test string (* if the string exists (i.e., is defined) *)
if test command (* if the command is ok *)

b) test {-r -w -f -d -s} file
(* check whether file is readable, writable, not a directory, a directory, exists and length >0 *)
EX. if test -f $1

c) test value1 operator value2
(* operator ={-eq -ne -gt -lt -ge -le} *)

```
if test $# -ne 2
then
    echo error
    exit 1
fi
```

d) and/or/not ==> -a/-o/!

```
if test \( -r fil1 -a -w fil1 \) -o \( -r file2 -a    -w file2 \)
```

e) if test cond ==> if [cond]

```
if [ $age -le 6    -a $age -gt 2 ]
```

* Remarks for writing a shell program

1) check the number of parameters.
2) if the parameter should be a file, check whether it exists.
3) if the file should be readable, check whether it is.
4) otherwise, print error messages.

* trap --> exception handling

```
trap command signal_list
(* when signal_list occurs, execute the command *)
(* signals are used for process communication, ex. ^C *)
(* signal: a facility for handling exception conditions
similar to software interrupts *)

# grepss file word1 word2 ...
file=$1
shift
temp1=/tmp/mm.$$ (* $$ returns a unique pid *)
temp2=/tmp/pp.$$
trap 'rm -f $temo1 $temp2 ; exit 1' 1 2 15
# 1"handup" 2"control-C" 15    ..
for word
do
   grep $word $file > $temo1
   cp $temp1 $temp2
   file=$temp2
done
cat $temp1
rm -f $temp1 $temp2
```

* Beeper

```
while true
do
   sleep 20    (* seconds *)
   echo ^GTime is up^G
done
```

* Expression: expr value1 op value 2

```
(* there is a space at the both sides of op *)

$ expr 4 + 3
7
$ expr 4 \* 3
12
$ expr 14 / 5    (* like div *)
2

$ expr 14 % 5
4
$ pp= `expr 5 + 3`
8
$ pp=`expr $pp - 2`    (* like pp := pp - 2 *)
6
```

* String concatenation

```
pp=' '
while ..
do
   read name
   pp=$pp$name' '
done
```

* while .. if .. then break fi .. done
  while .. if .. then continue fi .. done

* bad: shell is not suitable for math. computation.
  bad: shell is not suit for some string operations.

* man command (* = help *)

特殊的 SHELL 字元和符號

通配字元的代換（檔名）

?	任何一個字元
*	任何字元的組合(包括零值字元)，而以句點開頭的組合除外
[list]	任何一個包括在 list 裡的字元
[^list]	任何一個不包括在 list 裡的字元

重導和管道

>file	使 file 成為標準輸出
<file	使 file 成為標準輸入
>>file	使 file 成為標準輸出，若 file 已存在，則把資料附加在它的後面
<<word	獲取 shell 的輸入資料，直到遇到第一個包含 word 的資料列或直到檔案終了為止
n>file	使 file 成為檔案描述值 n 的輸出；此處 0 表標準輸入、1 表標準輸出及 2 表標準錯誤
1>&2	把標準輸出重導到標準錯誤，command1|command2 使 command1 的標準輸出成為 command2 的標準輸入

其他命令列符號

\	命令延續到下一列
&	在後庭處理中執行前一命令
;	命令的分隔號
spaces,tabs,and newlines	字語的分隔號
( )	命令集合
command1 && command2	若 command1 成功，則執行 command2
command1 || command2	若 command1 失敗，則執行 command2

Shell 命令

:	虛擬命令；不做任何事，但可附有一些引數
.file	執行 file 裡的命令，但不會產生新的 shell
$#	位置參數的總數
$-	shell 選擇項
$?	最後一個被執行之命令的離開狀態
$$	目前 shell 的程式號碼（PID）
$!	最後一個後庭處理命令的程序號碼
$0	一個正被執行的命令名稱
$*	一系列位置參數（shell 的引數）
$@	與$*同，除了當被雙引號括起來時的作用不同外

引用

\	把跟在其後的字元逐字解譯
''	把括在引號內的所有字元逐字解譯
""	把括在引號內的所有字元逐字解譯，但 $ 的組合和* 則不加以逐字解譯，此兩者會被擴展成它們所對應的參數與檔名值

命令代換

`command`	把此運算是代換成反引數內 command 的輸出結果
#	把此符號後面所跟著的資料列解釋成註解

參數代換

${parameter-word}	若 parameter 已有值，則此運算式值為 parameter 的值：否則其值為 word
${parameter=word}	若 parameter 已有值，則此運算式值為 parameter 的值：否則其值為 word，且把 parameter 的值設定為 word
${parameter+word}	若 parameter 已有值，則把 parameter 的值換成 word，且此運算式的值為 word；否則運算式 parameter 均無值
${parameter?mesg}	若 parameter 已有值，則此運算式的值即 parameter 的值；否則其值會換成 mesg，且 mesg 一印出後就立刻離開 shell

```
#speller -- makes spelling changes      Version 1.2
#Usage: speller filename
case $# in
  1)  ;;
  *)  echo Usage: $0 filename 1>&2 ; exit 1 ;;
esac
if test ! -r "$1"
then
     echo $0: cannot read $1 1>&2
     exit 1
fi
missp=/tmp/sp.m.$$
chsp=/tmp/sp.c.$$
nouse=no_use.$$
spell "$1" > $missp
if test -r $missp -a ! -s $missp
then
     cat "$1"
     exit 0
fi
for word in `cat $missp`
do
   echo    The unidentified word is $word. > /dev/tty
   grep -n '^' "$word" '[^a-zA-Z]' "$1" > /dev/tty
   grep -n     '[^A-Za-z]' "$word" '[^a-zA-Z]' "$1" > /dev/tty
   grep -n     '[^A-Za-z]' "$word" '$' "$1" > /dev/tty
   echo Enter the correct word. To skip, hit [return] > /dev/tty
   read correct
   if    test "$correct"
   then
     echo 's/^' "$word" '\([^a-zA-Z]\)/' "$correct" '\1/
          s/\([^A-Za-z]\)' "$word" '\([^a-zA-Z]\)/\1' "$correct" '\2/g
          s/\([^A-Za-z]\)' "$word" '$/\1' "$correct" '/g'
```

```
   else          #save unmodified words
      echo $word >> $nouse
   fi
done > $chsp
sed -f "$chsp" "$1"
if test -r $nouse
then
   echo Unused words are in the file $nouse > /dev/tty
fi
rm -f $missp $chsp
```