Lecture 9

Shell Programming –

Command substitution

Regular expressions and grep

Use of exit, for loop and expr commands

COP 3353 Introduction to UNIX

Command Substitution

- a string in back quotes ` … ` does command substitution
  - This means that the result of the command (the standard output of the command) replaces the back quoted string

Examples:

```
count=`wc -w <$1`
```

# the value of count is assigned the number of words in file $1

```
if [ `wc -l < $2.txt` -lt 1000 ];
```

#checks if the number of lines in the file is < 1000

```
cat `grep -l exit *.sh`
```

#print out all *.sh files containing the word exit

Exit Command (again)

- Conventionally, zero normally indicates success. Nonzero values indicate some type of failure. It is thus good practice to ensure that if the shell script terminates properly, it is with an “exit 0” command.
- If the shell script terminates with some error that would be useful to a calling program, terminate with an “exit 1” or other nonzero condition.
- most Unix utilities that are written in C will also call “exit(<value>);” upon termination to pass a value back to the shell or utility that called that utility.

Exit Example

- The following shell script exits properly. It also distinguishes the response through the value returned.

```
#!/bin/sh
#determines a yes (0) or no (1) answer from user
echo “Please answer yes or no”; read answer
while :
do
    case $answer in
    “yes”) exit 0;;
    “no”) exit 1;;
    *) echo “Invalid; enter yes or no only”
        read answer;;
    esac
done
```

Testing the Exit Status

- Conditions tested in control statements can also be the exit status of commands. Assume that the script “yes.sh” has been invoked.
- The following segment will test this as part of its script:

```
if yes.sh
then
    echo “enter file name”
    read file
else
    echo “goodbye”; exit 0
fi
```

Regular Expressions and Wildcards

- Many Unix utilities use regular expressions
- A regular expression is a compact representation of a set of strings
- Note that the shell uses wildcards (*, ?, etc.) for filename matching. The special characters are not necessarily used the same way in regular expressions
- Thus the pattern “alpha*.c” for filenames is not the same when used in the grep command (for example) to match a regular expression!
- In a regular expression, “*” means match zero or more of the preceding character

Regular expression operators

- Concatenation
  - This is implicit and is simply one character followed by another.

```
ab      #matches the character "a" followed by "b"
alpha   #several characters concatenated
```

- * operator:
  - indicates zero or more instances of the preceding character or preceding regular expression if grouping, that is parentheses (,), are used.

```
ab*c    #matches ac, abc, abbc, etc.
```

- + operator:
  - similar to * except matches 1 or more instances of the preceding character

Matching a specific class of characters

- "." matches any single character except newline

  a.b #matches a followed by any character, then b

  # for example adb, a&b, etc.

- [] is used to indicate one of a set of characters. The '-' is used to define a range. A "^" after "[" means match anything not in the set.

  [adkr] #match a, d, k, r

  [0-9] #match 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

  [a-z] #match lower case letters

  [^aeiou] #match any character except a vowel

  [^0-9] #match any character except a decimal digit

Anchors

- Anchors ^ and $ can be used to indicate that a pattern will only match when it is at the beginning or end of a line (note that the following use of “^” is different from its use inside a set of characters)

```
^alpha      #match the string “alpha” only when it
            #is at the beginning of the line
[A-Za-z]+$      # a name at the end of the line
^alpha*zeta$  #start with alph, end with zeta and
                #any number of “a”s in between
```

Alternation and Grouping

- Use the “|” character to choose between alternatives. Parentheses are for grouping

  a|b		#match a or b

  a*|b	#match any number of a’s or a b.

  (ab*a)*	#any number of ab*a

grep and egrep

- grep searches for strings in files that match a regular expression and prints out the lines that contain these matches to stdout. If no file is specified then grep uses stdin.
- form (the initial brackets indicate some optional flags):

```
grep [-i] [-w] [-c] [-v] [-E] pattern [files]
```

- egrep is extended grep and extends the syntax of regular expressions. Generally grep does not support the parentheses, the + operator, the | operator or the ? operator (zero or one occurrence). The flag –E in grep generally gives egrep behavior.

Grep options

- -i will make the search case insensitive
- -c will cause the number of lines matched to be printed
- -w will force the search to look for entire words
- -v will cause the lines that *do not match* to be output
- -l will return only the name of the file when grep finds a match

Grep examples

```
grep alpha junk         #look for the substring alpha in file junk

grep “ii*” junk         #look for the substring of one or more i’s

grep ^begin junk        #look for a line that starts with begin

grep recieve *.sh       #find a “recieve” in any file ending in .sh

grep “[abc].*” junk     #find a substring with an a, b, or c, followed
                        #by any number of other characters
```

For statement

- The shell <variable> is assigned each word in the list, where the set of commands is performed each time the word is assigned to the variable. If the “in <word_list>” is omitted, then the variable is assigned each of the command line arguments.

```
for <variable> [ in <word_list> ]
do
one or more commands
done
```

For statement examples

```
#!/bin/sh
#makes a backup of certain files and echoes arguments
for file in `ls *.c`
do
    cp $file $file.bak
done
for arg
do
echo $arg
done
exit
```

Examples cont

```
#!/bin/sh
#place command line arguments into a string variable
#arguments separated by a blank space
s=“”
for arg
do
    s=“$s $arg”
done
#compile each of the files in the list $s
for file in $s
do
    gcc –g –c #file
done
exit 0
```

Expr

- expr evaluates an arithmetic or relational expression and prints its result to standard output. This is useful when you need to perform calculations in the shell script. It outputs 1 (true) or 0 (false) when evaluating a relational expression.
- Note that the arguments are operators must be separated by spaces.
- Example from the tcsh command line (note *set*)

```
set alpha = 3
expr $alpha + 2 #result printed out is 5
```

More expr examples

```
var=`expr $var + 1`                  #increment var by 1

if [ `expr $s1 \< $s2` = 1 ]         #check if the value of  s1
                                     #is less than value of s2

beta=`expr $beta \* 2`               #multiply value of beta by 2
set beta = 10; expr $beta / 2        #using tcsh directly, result is 5

expr “$alpha” = hello                #output 1 if variable alpha is
                                     #hello
```