**Unix and C Programming 04**

**Scripting Part 2**

The directory /var/log/httpd contains the log files for the Sciris web server program. There are several files names access\_log\* and error\_log\*.

1. Display the first few entries in that file. Use a web search to find out what the entries in each line mean. As a hint, the Apache Web server on Sciris uses the “Combined Log Format”.
2. Find out if anyone has accessed your web page on Sciris lately. How many times? If nobody accessed your pages, how about mine (<http://sciris.shu.edu/~wachsmut>)
3. Note that the access files “rotate” weekly, i.e. access\_log is copied to access\_log.1, access\_log.1 to access\_log.2, etc, and the oldest file is removed. How would you display all access counts of your or mine, or any user’s web page during the past 4 weeks?

The first questions we should be able to do without trouble. For the third we need some more programming background. Here are the basics on more advanced shell programming. We do not need all features to answer the third question, but we do need some. Which ones is left to you; we’ll pick up the example later and solve it.

**Passing arguments to the shell**

Shell scripts can act like standard UNIX commands and take arguments from the command line. Arguments are passed from the command line into a shell program using the positional parameters $1 through to $9. Each parameter corresponds to the position of the argument on the command line. The positional parameter $0 refers to the command name or name of the executable file containing the shell script. All the positional parameters can be referred to using the special parameter $\*. Only nine command line arguments can be accessed directly, but you can access more than nine [using the shift](http://unixhelp.ed.ac.uk/scrpt/scrpt2.1.3.html) command (use the manual pages for details).

echo “The first five arguments are $1 $2 $3 $4 $5”

echo “The command was $0”

echo “All parameters were $\* anad there were $#”

**Reading user input**

To read [standard input](http://unixhelp.ed.ac.uk/glossary/gs.html#stdin) into a shell script use the read command. For example:

 echo "Please enter your name:"

 read name

 echo "Welcome to South Orange, $name"

This prompts the user for input, assigns this to the variable name and then displays the value of this variable to [standard output](http://unixhelp.ed.ac.uk/glossary/gs.html#stdout). If there is more than one word in the input, each word can be assigned to a different variable. Any words left over are assigned to the last named variable. For example:

 echo –e "Please enter your surname followed by your first name: \c"

 read name1 name2

 echo "Welcome to SHU, $name2 $name1"

**Computations**

The shell does not offer any arithmetic support. You need to use the command expr instead. For example

 echo “6 + 6 = `expr 6 + 6`”

For details on this command, use the manual pages.

**Conditional statements**

Every Unix command returns a value on exit which the shell can interrogate. This value is held in the read-only shell variable $?. You can also query the exit value directly with, for example, an if statement.

The if statement uses the exit status of the given command and conditionally executes the statements following. The general syntax is:

 if test

 then

 commands (if condition is true)

 else

 commands (if condition is false)

 fi

then, else and fi are *shell reserved words and as such are only recognised after a newline* or ; (semicolon). Make sure that you end each if construct with a fi statement.

if statements may be nested:

 if ...

 then ...

 else if ...

 ...

 fi

 fi

The elif statement can be used as shorthand for an else if statement. For example:

 if ...

 then ...

 elif ...

 ...

 fi

For example:

if who | grep -s keith > /dev/null

 then

 echo keith is logged in

 else

 echo keith not available

 fi

Note the usage of /dev/null to suppress the output of the grep command by redirecting it to “null” (a digital garbage can).

**Testing for files and variables with the test command**

The shell uses a command called **test** to evaluate conditional expressions. For example:

 if test ! -f $FILE

 then

 if test "$WARN" = "yes"

 then

 echo "$FILE does not exist"

 fi

 fi

First, we test to see if the filename specified by the variable $FILE exists and is a regular file. If it does not then we test to see if the variable $WARN is assigned the value yes, and if it is a message that the filename does not exist is displayed. Full details of this command can be found in the [test](http://unixhelp.ed.ac.uk/CGI/man-cgi?test) manual page.

**The for statement**

The for loop notation has the general form:

 for var in list-of-words

 do

 commands

 done

where commands is a sequence of one or more commands separated by a newline or ; (semicolon). The *reserved words do and done must be preceded by a newline* or ; (semicolon). Small loops can be written on a single line. For example:

 for var in list; do commands; done

Example:

 #!/bin/sh

 # See if a number of people given on the command line are logged in

 for i in $\*

 do

 if who | grep -s $i > /dev/null

 then

 echo $i is logged in

 else

 echo $i not available

 fi

 done

**The while and until statements**

The while statement has the general form:

 while command-list1

 do

 command-list2

 done

The commands in command-list1 are executed and if the exit status of the *last* command in that list is 0 (zero aka “true”), the commands in command-list2 are executed. The sequence is repeated as long as the exit status of command-list1 is 0 (zero). The until statement , on the other hand, has the general form:

 until command-list1

 do

 command-list2

 done

This is identical in function to the while command except that the loop is executed as long as the exit status of command-list1 is non-zero. For example, to get notified when a file has been created:

 #!/bin/sh

 until test -f $FILE

 do

 sleep 60

 done

 echo "$FILE now exists"

This tests every 60 seconds until the filename represented by the variable $FILE exists. A message is then displayed. Exercise: Write a shell script that display a message if a file gets deleted.

Now we are ready for our original 3rd problem: *How would you display all access counts of your (or mine) web page during the past 4 weeks?*

* First create a command to count the number of accesses in “/var/log/httpd/access\_log” and test it
* Next create a script to loop through all files starting with access\_log and display the file name to test
* Combine the previous two approaches to list counts for all access\_log files
* Use command line input for the user name to check, and use variables for easy configuration
* As a bonus, use a counting variable and the *expr* command to add up the counts (this would be the trickiest part).

*How to display all access counts of your or mine or any user’s web page during the past 4 weeks?*

#!/bin/sh

#

# Date: Feb 2011

# Author: Bert Wachsmuth

# Counts the total number of access for a personal web page on Sciris

# Name of httpd log files (this should be the shared part of the name if there are several)

LOG\_FILE="/var/log/httpd/access\_log"

# Display brief description

echo -e "$0: computes total number of hits for user's personal web page on $HOSTNAME\n"

# Checking if there are enough input params

if test $# -lt 1

then

 echo -e "Usage:\n\t $0 NAME, where NAME is a username\n"

 exit -1

else

 NAME=$1

fi

# Checking if access log file can be located

if test ! -f ${LOG\_FILE}

then

 echo -e "Error. Log file ${LOGDIR}${LOG\_FILE} not found.\n"

 exit -1

fi

# Now counting how many times NAME appears across all access log files

count=0

for file in `ls ${LOG\_FILE}\*`

do

 x=`grep -c $NAME $file`

 echo "$x counts of $NAME in $file"

 count=`expr $x + $count`

done

echo "Total count of $NAME is $count"