Ken’s Perl Notes

Contents

Variables

$ = scalar variable (string or numeric)
@ = indexed array or list
% = hashed or associative array
& = function or subroutine

Lists and Arrays

To store a list of an array (or a hash), you need an array variable which begins with a "@"

```
@a = (5, 'apple', $x, 3.1415);     will perform variable substitution
            $a[0]                  is 5
@b = qw(5 apple $x 3.1415);    wont do variable substitution
@c = <STDIN>                       reads all input in
$d = <STDIN>                       reads one line in
```

Arrays are numbered starting at index 0

Push (or unshift) and pop (or shift)

```
@band=qw(trombone);
push @band, qw(ukulele clarinet);
# band now contains trombone, ukulele, clarinet
$brass=pop @band;   # brass now has trombone
```

```
$anArray = [6,7,8,9,10,11,12,13,14,15];  # ref to an anonymous array
# qw : returns a list of whitespace delimited words
$anArray2 = ( qw [blue_shirt hat jacket preserver unscreen] );

$anArray->[0];         # deref first element [0]
$$anArray[0];          # another way to do

my $pArray = \@argArray;  # pointer to an array
$aSize = $#$pArray;         # tricky! – give max array index
```

Splice (somewhat like Tcl lindex)

removes elements from *array* starting at *offset* , and the removed elements are returned

splice *array*, *offset*
splice *array*, *offset*, *length (or number of elements to remove)*
splice *array*, *offset*, *length*, *list(the removed elements get replaced with the contents of this list)*

```
@veg=qw( carrots corn );
splice(@veg, 0, 1);   # @veg is carrots
splice(@veg, 0, 0, qw(peas));   # @veg is peas, corn
```

@c = ('john', harry', ('mike, 'jim'), 'susan');   *will flatten the structure (I thnk)*

$size_of_c = @c *number of elements in @c* scalar(@c) *also does this*
$last_index = $#c *index of last element in @c*

@_ # subroutine arguments
$_[0] # first argument passed into a subroutine - has nothing to do with global scalar $_

foreach $item (@reply) { *just like in Tcl*

if (@c) { *tests whether there's anything in @c*

for ($i = 0; $i < @c; $i++) { *another example*

($a, $b, $c) = qw(apple pear orange) *Tcl lset-like*

($a, $b, $c, $d) = qw(apple pear orange) *$d doesnt get a value*

print scalar(localtime) *returns a nicely formatted string*
($sec, $min, $hour, $mday, $mon, $year, $wday, $ydat, $isdst) = localtime *array init*

Hash

(Perl hashes are really indexed arrays) begin with "%"

Inputting:

```
$food{'apple'} = 'fruit';
$food{'carrot'} = 'vegetable';
        or
%food = ('apple','fruit','carrot','vegetable');
        or
%food = ('apple' => 'fruit', 'carrot' => 'vegetable');
```

Outputting:

```
$whatType = $food{'carrot'};

foreach $film (keys %Movies) {
   print "$file\n";
}

if  ( exists $Movies{'Patton'} ) {

}
```

Deleting

```
delete $Movies{'Patton'};
```

```
$aHash = {
    name => 'ken freed',
    age  => 54,
    weight => 235,
    wife => 'vannessa'
    };                        # initializing an anonymous hash

$aHash->{wife};               #deref
$$aHash{wife};                # another way to deref

$pArgHash  = \%anotherHash;     # pointer to a hash
```

Common Hash Data Structure

```
my $gSEM15 = {
               ip => 'sem15.mycompany.local',
               logon => 'xyzuser',
               pw => 'xyzuser1',
               srce_dir => '/reports/Production/',
               dest_dir => 'c:/sem15_incoming'
             };

my $gSEM17 = {
               ip => 'sem17.mycompany.local',
               logon => 'xyzuser',
               pw => 'xyzuser1',
               srce_dir => '/reports/Production/',
               dest_dir => 'c:/sem17_incoming'
             };

my @gTOOLS = ( \$gSEM15, \$gSEM17 );

foreach my $aTool (@gTOOLS) {
        $aValue = $$aTool->{ip};
}
```

More Terse Hash

```
my @gTOOLS = (
               {
                 ip => 'sem15.mycompany.local',
                 logon => 'xyzuser',
                 pw => 'xyzuser1',
                 srce_dir => '/reports/Production/',
                 dest_dir => 'c:/sem15_incoming'
               },

               {
                 ip => 'sem17.mycompany.local',
                 logon => 'xyzuser',
                 pw => 'xyzuser1',
                 srce_dir => '/reports/Production/',
                 dest_dir => 'c:/sem17_incoming'
               }
            );

foreach my $aTool (@gTOOLS) {
        $aValue = $aTool->{ip};
}
```

Strings

IMPORTANT: use eq instead of == for string compares, e.g.:

```
    if (lc($pieces[0]) eq 'lot') {   # RIGHT
    if (lc($pieces[0]) ==  'lot') {   #WRONG
```

Embedding quotes in strings: “The I said to him \”Go ahead, make my day.\””

qq(Then I said to him, “Go ahead, make my day”)
qq[Then…] *you can use any type of brackets*

Double quoted strings get variable replacement, single quoted strings do not.
q{Then I said to him… }

Concatenation $a = Ken
$b = Freed
$c = $a . $b
print “$a $b” ; *will also concatenate*

$line .= “make my day”; *appends to the line end*

$adrLine = \$line; # the address (or a reference to) $line

Repetition operator $line = “-“ x 70;

int(6.345) *returns the integer portion*
length(“KenFreed”)
lc(“KenFreed”) returns lower case
uc(“KenFreed”) returns upper case
cos(50)
rand(5) random number from 0 to less than its argument
$a++ *just like in C*

Text Input (scanf like) print “What is your name?”;
$name=<STDIN>;
print “Your name is $name”;

SUBSTR

substr *string, offset*	*returns the rest of the string, starting at offset*
substr *string, offset, length*	*the first charcacter is at offset zero*

$a = "I do not like green eggs and ham"
substr($a, -5); # returns the last 5 chars of $a
substr($a, 5); # first 5 chars
substr ($a, 5, -10); # 5 chars, starting 10 chars in from the end

using substr on the left:
$a="countrymen, lend me your wallets";
substr($a, 0, 1) = "Romans, C"; # replaces the fist char w/the string on the right
substr($a, 0, 0) = "Friends, "; # insert Friends at the start
substr($a, -7, 7)="ears."; #replaces the last 7 chars with these 5

printf, sprintf

Formatting: c *character* s *string* d *integer* f *float*

```
printf("%20s", "Jack");     # prints Jack  in  20 chars
printf("the amount is %6d", $amount);
```

use English; # enables better names for the special vars

$_	$ARG
@_	@ARG
$!	$OS_ERROR
$^O	$OS_NAME
$O	$PROGRAM_NAME

Sending out email from a Perl script:

```
system("mailx -s $subject_line $address_string < $trackreport");
```

SPLIT

@words=split(/ /, "The quick brown fox"); *uses the blank between the slashes as delimiter*
' ' is thre null charcter - splits into individual chars

```
@line_pieces  = split (/\s+/, $aLine);     # split on whitespace
```

$_ is the global which is split, if no string is specd

```
while (<STDIN>) {
        @a = split(//, $_)
}
```

JOIN

@presidents=(Clinton Bush Reagan Carter Ford Nixon);
print join (' ', sort @presidents) *prints out in ascii sorting order:* Bush Carter ...

print join(' ', reverse sort @ presidents) *decending order*

INDEX

```
index "ring around the posey", "around";   # returns 5

$source = "one fish two fish red fish blue fish";
$start=-1;
while (($start=index($source, "fish, $start)) != -1) {
        print "found a fish at $start\n";
        $start++;
}
```

rindex searches from right to left

Match

if ($line =~ "Recipe:") {   #contains Recipe:

if ($input_option =~ /^add/) {   # starts with add

if ($device_name =~ /^[rR]$/) { # start and ends with r in either case

Transliteration (substitution)

$action =~ tr/[a-z]/[A-Z]/; # make capitals

Datetime

use Time::localtime;

scalar(localtime)

gives (e.g.):

Tue Mar 13 09:47:17 2007

my($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst) = localtime(time);

- $year is the number of years since 1900, not just the last two digits of the year. That is, $year is 123 in year 2023. The proper way to get a 4-digit year is simply: $year + 1900
- To get the last two digits of the year (e.g., '01' in 2001) do:

```
$year = sprintf("%02d", $year % 100);
```

- $mon is the month itself, in the range 0..11 with 0 indicating January and 11 indicating December. This makes it easy to get a month name from a list:

($sec, $min, $hour, $mday, $month, $year) = (localtime)[0, 1, 2, 3, 4, 5];

```
    # Thu       Nov       30   03:50:01 2006
 my ($wday, $month, $day, $time, $year) =  split (/\s+/, scalar(localtime));

 my ($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdist) = localtime();

 my $tm = timelocal($sec, $min, $hour, $mday, $mon, $year);
```

- $wday is the day of the week, with 0 indicating Sunday and 3 indicating Wednesday.

```
my @gWEEKDAY = ('??','Mo','Tu','We','Th','Fr','Sa','Su');
 my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime();
 return $gWEEKDAY[$wday] . " " . "$hour:" . sprintf("%02d",$min);
```

Looping, Iteration, Decisions

FOR

```
        for ($i = 0; $i < 15; $i++) {

        }
```

IF THEN ELSE WHILE ...

Relational operators = = , ! = but use*:* eq, ne, gt, lt,… for strings
Logical Operators && *or* and, || *or* or, …. *just like in C*

The number 0, the empty string, an undef value *are false*
everything else is true
0.0 is false, but “0.0” is true
“0.0” + 0 is false, evaluates as 0

logical expressions return the last value evaluated
5 && 7 returns 7

```
$new = $old || “default”;
```

is the same as:

```
$new = $old;
if (! $old) {
   $new = “default”
}
```

if, for - are just like in C
but there's another if form:

```
if (test-expression) {
statement;
}
```

is the same as

```
expression if (test-expression);
```

the “last” statement exists an innermost loop

```
while ($i  < 15) {
last if ($I ++ 5);
$i++;
}
```

can also put a label on the “last” - for existing other loops

```
OUTER:  for ($i  = 0; $i  < 100; $i++) {

    last OUTER;
```

spaceship operator { $a <=> $b } *returns -1 if $a < $b, 0 if $a = $b, +1 if $a > $b*

my

```
while (my $line=<STDIN>)     {   # $line is local to this while loop
foreach my $item (@array)      {    # $item is local to this loop
while (local($line)=<STDIN>) {    # local( ) is the same as my
```

tricky:

```
foreach (@data) {
        sum += $_;     # tricky! - makes use of the global $_ as the item
}
```

Subroutines

```
sub afunction  {
            my($parm1, $parm2) =  @_;   # stack copies
            $first_arg = $_[0];                 # to get the first arg, original, not a stack copy
            return ($someValue);
}
```

different calling styles:

```
            afunction arg1, arg2, arg3;
            afunction(arg1, arg2, arg3);
```

&afunction(arg1, arg2, arg3);   # use the ampersand in front if the call is before the function declaration

This

If a subroutine is called in an OO style, the $this (or $class) is automatically the first parm

```
            sub aSub {
              my ( $class, $q, $addr ) = @_;

            aClassInstance->aSub($a, $addr);
```

Initializing Subroutine Hash Args

```
sub aSubroutine {

  my $this = shift;  # under OO, this gets passed as first arg automatically

  my %argHash = @_;  # convert the list to a hash

# various way to init hash passed in
  $argHash{pageSize} = 80 unless exists $argHash{pageSize};

# or another way
     %argHash =( name => $arg{name} || "???" );

# which is the same as
     %argHash =( name => defined($arg{name}) ? $arg{name} : "???"  );

# or another way
  my %defaults = (cols => 8, rows => 20, bLogit => 0, pageSize => 80);

     %argHash = (%defaults, @_);
```

wantarray

```
  # What the does caller expect to be returned? =
  # What is the caller's context?
  #--------------------------------------------------
  if ( wantarray() ) {             # true
    print ("\naSubroutine caller expects an array or hash to be returned");
  } elsif (defined wantarray() ) {  # false but defined
      print ("\naSubroutine caller expects a scalar to be returned");
  } else {                          # false and undefined
      print ("\naSubroutine does not expect a return (void)");
  }
```

OO

Subroutine example

```
sub prior_customer {

  my $this = shift;  # class is first parm in OO style -> call

  my %defaults = (db            => undef,
                  subbrand      => 'giganews',
                  email         => undef,
                  card          => undef,
                  skip_deleted_before => 0,   # if undef than we are not skipping anything
                  skip_trials       => 0);

  my %args = (%defaults, @_);
```

Calling Example

```
# prior accounts, since the start of giganews
  my %argHash = (db            => $gndb,
                 subbrands    => undef,
                 email        => $q->param('outside_email'),
                 card         => $q->param('pay_data1'),
                 deletedAfter => scalar(parsedate('01/01/1998')),   # founded in 1998
                 entryAfter   => undef,
                 bOnlyCountUnlocked => 0,
                 bOnlyCountTrials   => 0);

  my $accounts = GN::Web::Signup->prior_customer( %argHash );  # class is automatically 1st arg

  $argHash{bOnlyCountTrials} = 1;

  my $trials = GN::Web::Signup->prior_customer( %argHash );
```

Transliteration, Regular Expressions

tr/*searchlis/replacementlist/*
y/*searchlis/replacementlist/* *# y is an old symonym for tr*

tr/ABC/XYZ/; # In $_ replaces all As with Xs, Bs with Ys, etc.
$r=~tr/ABC/xyz/; # same as above, only does this for $r; =~ is the "binding operator"

tr/A-Z/a-z/; # change all upper case to lower case
tr/A-Za-z/a-zA-Z/; # invert upper and lower case

if replacement list is empty (or the same as searchlist), the searchlist chars are counted

$eyes=$potatoe=~tr/i//; # count the i's in $potato, return number to $eyes

tr(a-z)(n-za-m); # rotate all chars to the left in $_

Common Regular Expressions

```
$parm =~ s/^\s+//;    # remove leading spaces if any
$parm =~ s/\s+$//;    # remove trailing spaces if any
if ( $aLot =~ m/2[0-9][0-9][0-9][0-9][0-9][0-9]/ ) {  # sanity check on lot
} elsif ($line =~ "8200-") {
$wfr =~ s/^0//;    # remove 1st char to make 2 digit wafer #
$wfr =~ s/^/0/;    # add first zero
$part_device =~ s/^..//;  # remove first two chars '7C'
$part_device =~ s/.$// ;  # remove last character   'C'
if (! $gbCONFIG_USING_UNIX) { # substitute windows backslash for unix forward slash
    $backupFile =~ s#/#\\#g;
}
$probecard =~ s/^\D*0*([0-9]*)\D*$/$1/;  # Extract only the digits, no leading zeroes.

# now get the 3-6 numbers, followed by a letter
    m/(\d{3,6})(\D)/;
    my $aNumber = $1;
    my $aLetter = uc($2);
```

warn "has nondigits" if /\D/; warn "not a natural number" unless /^\d+$/; # rejects -3

warn "not an integer" unless /^-?\d+$/; # rejects +3 warn "not an integer" unless /^[+-]?\d+$/;
warn "not a decimal number" unless /^-?\d+\.?\d*$/; # rejects .2
warn "not a decimal number" unless /^-?(?:\d+(?:\.\d*)?|\.\d+)$/;
warn "not a C float" unless /^([+-]?)(?=\d|\.\d)\d*(\.\d*)?([Ee]([+-]?\d+))?$/;

File IO

open(aHANDLE, "myfile.txt") || print "Cannot open myfile.txt"
if the left hand side evaluates to false, the right must then be evaluated

```
                    open(aHANDLE, ”<”, ”c:\\kendoc\\myfile.txt”)    # open for reading
```

when you use a backslash in a double quoted string...

```
                    open(aHANDLE, "c:\\kendoc\\myfile.txt")
                    open(aHANDLE,  "c:/kendoc/myfile.txt")    unix slashes will also work for Win
                    open(aHANDLE, ">>c:/kendoc/myfile.txt"); # open for appending
                    open(aHANDLE, ‘>>’, ‘c:/kendoc/myfile.txt’); # 3 parm form – more web secure
                    open(aHANDLE, ">c:/kendoc/myfile.txt"); # open for writing, dont preserve old data
                    print aHANDLE "some data";   # to write to the file
```

writing trick:

```
                    open (aHANDLE, ">>logfile.txt") || die "$!";
                    if (! print aHANDLE "scalar(localtime), here is a line to write in the file\n") {
                              warn "unable to write to the log file"   # the print returned TRUE if successful
                    }
                    close(aHANDLE);
```

binmode(aHANDLE); # now we treat it as a binary file
seek (*filehandle, offset in the file, relative to 0=file beginning | 1=current position | 2=end of file);*

"die" function - gracefully stops the interpreter and prints an error message

```
                                        open(aHANDLE, "myfile.txt") || die "Cannot open myfile.txt: $!\n"
```

note the use of the global "$!" above - it has the error text

there's a "warn" function too - only the interpreter keeps going

```
$line = <aHANDLE>;           # reads a line from the file
@contents = <aHANDLE>;   # reads the entire file

while (defined($a=<aHANDLE>)) {         # to read the whole file, line by line

}

while (<aHANDLE>) {
          print chomp($_);                  # if angle operators are only ones, then read is assigned to global $_
}                                          # chomp gets rid of the end of line chars (LF for unix, CRLF for win)
```

File test operators

```
                              print "Save data to what file?";
                              $filename=<STDIN>;
                              chomp $filename;
                              if (-s $filename) {
                                        warn "file contents will be overwritten\n"
                                        warn "file was last updated", -M $filename, " days ago.\n";
                              }
```

Directory IO

opendir *dirhandle, directory;*

```
opendir(TEMPDIR, '/tmp') || die "Cannot open /tmp: $!";
```

readir *dirhandle;*

```
@files = readdir TEMPDIR
@files=grep(/\.txt/i, readdir TEMPDIR);  # elim the . and ..
```

closedir *dirhandle;*

mkdir *newdir, permissions; # permissions are a unix thing*
rmdir *pathname;*
unlink *list_of_files (or $_);* # to remove files from a directory, returns number of files removed.

```
$erased = unlink 'old.exe', 'a.out', 'personal.txt'
```

rename *oldname, newname;* # can also rename directories with this

```
rename "myfile.txt", "/tmp/myfile.txt";  # effectively moves the file
```

```
chdir '/tmp' or warn "Directory /tmp is not accessible: $!";
print "You are now in: ", cwd, "\n");
```

```
chmod 0755, 'file.pl'     # for unix systems
```

stat *filehandle;* # everything you ever wanted to know about it
stat *filename;*

globbing

```
my @hfiles=glob('/usr/include/*.h');
@cfiles=<*.c>;
```

- there's a limit to the number of files glob can return
- glob returns the full pathname whereas opendir... does not
- glob is slower than opendir...

Typeglobs and File Handles

The main use of typeglobs in modern Perl is create symbol table aliases. This assignment:

```
    *this = *that;
```

makes `$this` an alias for $that, `@this` an alias for @that, `%this` an alias for %that, `&this` an alias for &that, etc. Much safer is to use a reference. This:

```
    local *Here::blue = \$There::green;
```

temporarily makes $Here::blue an alias for $There::green, but doesn't make @Here::blue an alias for @There::green, or %Here::blue an alias for %There::green, etc. See Symbol Tables for more examples of this. Strange though this may seem, this is the basis for the whole module import/export system.

Another use for typeglobs is to to pass filehandles into a function or to create new filehandles. If you need to use a typeglob to save away a filehandle, do it this way:

```
    $fh = *STDOUT;
```

or perhaps as a real reference, like this:

```
    $fh = \*STDOUT;
```

See the perlsub manpage for examples of using these as indirect filehandles in functions.
Typeglobs are also a way to create a local filehandle using the `local()` operator. These last until their block is exited, but may be passed back. For example:

```
    sub newopen {
        my $path = shift;
        local *FH;  # not my!
        open   (FH, $path);          # or  return undef;
        return *FH;
    }
    $fh = newopen('/etc/passwd');
```

Now that we have the *foo{THING} notation, typeglobs aren't used as much for filehandle manipulations, although they're still needed to pass brand new file and directory handles into or out of functions. That's because *HANDLE{IO} only works if HANDLE has already been used as a handle. In other words, `*FH` can be used to create new symbol table entries, but *foo{THING} cannot.
Another way to create anonymous filehandles is with the IO::Handle module and its ilk. These modules have the advantage of not hiding different types of the same name during the `local().` See the bottom of open() for an example.

DBI

```
use DBI;

sub open_database_connection {
  my($dbname) = @_;
  my $hDB = 0;
  $hDB = DBI->connect($dbname);
  if ($hDB == undef) {
  // error
  }
  return $hDB;
}

sub close_database_connection {
  my ($hDB) = @_;
  $hDB->disconnect if ($hDB);
}

sub execute_sql {
  my ($hDB, $theSQL) = @_;
  my @returned_hashrefs = ();
  my $sth = 0;

  $sth = $hDB->prepare($theSQL);
  # these throw exceptions if table doesn’t exist
  eval {$sth->execute() } ;
  if ( $@ ) {
    // error
  } else {
    eval { while (defined(my $hashref=$sth->fetchrow_hashref())){
             push @returned_hashref, $hashref;
         }};
  }
  $sth->finish();
  return (@returned_hashrefs);
}

#caller
$theSQL = qq|SELECT DISTINCT * FROM aTable WHERE ID=56|;
@db_rows = execute_sql($hDB, $theSQL);
foreach (@db_rows) {
  push @some_collection, $_->{‘aColumn’};
}
```

Sql Dbi Injection

For web security. Often skipped over in the literature, using sql injection checks to make sure the user did not enter something like “drop table xyz” in their input.

On the execute statement

```
my $sth = $db->prepare_cached( q|
    SELECT c.lock_code, s.delete_date, s.flags
      FROM customer c, service s, service_attr sa
     WHERE s.customer=c.customer AND s.sid=sa.sid AND c.commercial=0 AND
           s.parent=1 AND sa.name in ( 'outside_email', 'login' )
           AND sa.data=? AND c.system=?
  | );
          sa.data

  $sth->execute( $email, $system );
```

Via Bind

```
    $sth = $db->prepare_cached( q|
      SELECT c.lock_code, s.delete_date, s.flags
        FROM customer c, service s
       WHERE s.customer=c.customer AND c.commercial=0 AND s.parent=1 AND
             c.pay_data3 = ? AND c.pay_data1 IN ( ?, ? ) AND c.system=?
    | );

    # make sure db driver treats these as varchars, not numbers, for indexing
    $sth->bind_param( 1, $last4, SQL_VARCHAR );
    $sth->bind_param( 2, $card, SQL_VARCHAR );
    $sth->bind_param( 3, $enc_card, SQL_VARCHAR );
    $sth->bind_param( 4, $system, SQL_VARCHAR );

    $sth->execute();
```

Common Perl Subroutines

```
my $gbCONFIG_USING_UNIX = 0;
#**************************************************
# logfile
#**************************************************
my $gLOGFILE_NAME = "c:/somedir/pgm_name.log";
my $gLAST_LOGFILE_NAME = "c:/somedir/pgm_name.lo1";
my $gMAXLOGFILESIZE = 128000;

#*****************************************************
# my_trim
#
# remove any leading and trailing whitespace from the
# parm passed in, then pass it back out
#*****************************************************
sub my_trim {
    my @theArgs = @_;
    my $parm = $theArgs[0];

    chomp ($parm);

    $parm =~ s/^\s+//;    # remove leading spaces if any
    $parm =~ s/\s+$//;    # remove trailing spaces if any

    return $parm;
}                        # my_trim

#****************************************************
# logmsg - log a message to the logfile if it is
#          open and OK, else just display the message
#          on STDOUT if not
#
#          uses file handle hLOGFILE
#****************************************************
sub logmsg {
    my ($theMsg) = @_;

    if ( length($theMsg) > 0 ) {

        my $theLine = scalar(localtime) . " " . sprintf($theMsg);
        my_trim($theLine);

        if (!(print hLOGFILE $theLine . "\n")) {  # if we cannot write it to disk
            print STDERR $theMsg . "\n";          # then just display it on the screen
        }

        print "$theMsg\n";

    }
}                                        # logmsg

#****************************************************
# logmsg_init - open and initialize the logfile
#****************************************************
sub logmsg_init {
    my $rc = 0;

#--------------------------------------------------------
# if our logfile has grown too big, copy it (overlaying)
# the backup file and shrink it back down to zero
#--------------------------------------------------------
    if ( (((stat $gLOGFILE_NAME))[7]) > $gMAXLOGFILESIZE ) {
        copy("$gLOGFILE_NAME",  "$gLAST_LOGFILE_NAME");   # uses File::Copy
        unlink $gLOGFILE_NAME;                            # deletes the file
        chmod 0666, "$gLAST_LOGFILE_NAME";
    }

    $rc = open (hLOGFILE, ">>$gLOGFILE_NAME");
    chmod 0666, "$gLOGFILE_NAME";
    return $rc;
}                                # logmsg_init


#******************************************************
# copy_file
#
# return 1=success, 0 otherwise
#******************************************************
sub copy_file {
    my ($sourceFile, $destFile) = @_;

    my $rc  = 1;
    my $cmd = "";

    $cmd = "copy $sourceFile $destFile";
    $cmd      =~ s#/#\\#g;    # unix forward slash to win backslash since we are using a
system cmd
    $destFile =~ s#/#\\#g;

    system ($cmd);         # we have to assume the system cmd executes OK

    if (! (-e $destFile ) ) {
        logmsg "ERROR $cmd";
        $rc = 0;
    }

    return $rc;

}                                 # copy_file


#**********************************************************************
# move_file
#**********************************************************************
sub move_file {
    my ($sourceFile, $destFile) = @_;

    my $cmd = "";

    if ( $gbCONFIG_USING_UNIX == 1 ) {
        $cmd = "mv \"$sourceFile\" \"$destFile\"";
    } else {
        $cmd = "copy \"$sourceFile\" \"$destFile\"";
        $cmd =~ s#/#\\#g;  # unix forward slash to win backslash since we are using a
system cmd
    }

    #if ( $gbDEBUG ) {
    #    print "$cmd\n";
    #}

    system ($cmd);                       # we have to assume the system cmd executes OK

    if ( $gbCONFIG_USING_UNIX == 0 ) {   # windows does not have a move, so delete after
the copy
        my $rc = unlink "$sourceFile";

        if (! $rc ) {
            logmsg "ERROR could not unlink \"$sourceFile\" (in move_file): $!";
        }
    }
}                                    # move_file
```