1. Download perl from http://www.perl.com/
2. What is Perl?

   Perl, which stands for "Practical Extraction and Report Language", was written by Larry Wall. Its a great language for doing data manipulation tasks. Its fast, portable, and accessible.

   If you toss a bunch of monkeys into a roomful of typewriters, the chances of them accidentally producing the complete works of Shakespeare are effectively nil. The chances of them producing a working Perl program, however, are actually good. -NICHOLAS PETRELEY, Computerworld JUN 03, 2002

3. File Association

   In windows XP the association of a .pl file needs to be something like:

   ```
   "D:\Projects\tools\i386\win32\bin\perl.exe" "%1" %*
   ```

   This is done in Explorer/View/Folder Options/File Types/Perl/open/Edit...

4. Some small programs to learn perl by example:
   1. HelloWorld.pl
   2. - Demonstrates,
      "#" is a comment,
      the "print" statement,
      "\n" is the linefeed character.

   ```
   3. #program HelloWorld.pl
   4. print "Hello World!\n";
   ```

   5. HelloWorld2.pl
   6. - Demonstrates:
      declaring a string variable on the fly
      printing a variable

   ```
   7. #program HelloWorld2.pl
   8. $name = "Sarah";
   9. print "Hello $name!\n";
   ```

   10. HelloWorld3.pl
   11. - Demonstrates:
       Interactively reading a string from the user
       the chomp() function

   ```
   12.    #program HelloWorld3.pl
   13.    # this demonstrates how to read from the user and use the
      chomp() function
   14.    print "What is your name?";
   15.    $name = <STDIN>;
   16.    chomp($name);   # get rid of the carriage return
   17.    print "Hello $name!\n";
   ```

   18. HelloWorld4.pl
   19. - Demonstrates:
       the equal operator, "eq", for strings
       if-else clause

   ```
   20.    #program HelloWorld4.pl
   21.    # this demonstrates the equal operator, "eq", for strings and
      if-else clause
   22.       print "What is your name?";
   23.       $name = <STDIN>;
   24.       chomp($name);   # get rid of the carriage return
   25.       if($name eq "Tex")
   ```

```
26.          {
27.           print "Howdy $name!\n";
28.          }
29.       else
30.          {
31.           print "Hello $name!\n";
32.          }
```

33. PrintArgs.pl
34. - Demonstrates:
    Reading arguments from the command line
    Using the standard argument array, @ARGV
    The for loop

```
35.    #program PrintArgs.pl
36.    # to print all the arguments passed:
37.         $numberOfArgs = @ARGV;
38.         print "The number of arguments passed was $numberOfArgs
   \n";
39.         for($i=0;$i < $numberOfArgs ; $i++)
40.         {
41.       print "argv[$i] = $ARGV[$i]\n";
42.         }
```

43. cat.pl
44. - Demonstrates:
    Reading from a file

```
45.    #program cat.pl, a tiny program to print a file.
46.    open(MYFILE,$ARGV[0]) || die "Cannot open file \"$ARGV[0]\"";
47.    while($line = <MYFILE>)
48.    {
49.        print "${line}";
50.    }
51.    close(MYFILE);
52.
53.
```

54. WriteToFile.pl
55. - Demonstrates:
    Writing to a file

```
56.    #program WriteToFile.pl, a tiny program to write stuff to a
   file.
57.    # "OUTFILE" is the name of the file handle
58.    open(OUTFILE,">test.dat") || die "Cannot open
   file \"test.dat\"";
59.    for($i=0; $i < 10; $i++)
60.    {
61.        print OUTFILE " line $i\n";
62.    }
63.    close(OUTFILE);
64.    print "Finished writing to \"test.dat\"";
```

65. paste.pl
66. - Demonstrates:
    Checking for the right number of arguments
    Opening multiple files
    exit statement

```
67.    #program  paste.pl, a tiny program to paste the lines of one
   file onto another.
68.    # ie, append to line1 of file1 with line1 of file2
69.    if( @ARGV < 2) # is less than two arguments
70.       {
71.          print "usage: paste.pl filename1 filename2";    exit 0;
72.       }
73.    # "FIRSTFILE" is the name of the file handle
```

```
74.    open(FIRSTFILE,$ARGV[0]) || die "Cannot open
   file \"$ARGV[0]\"";
75.    open(SECONDFILE,$ARGV[1]) || die "Cannot open
   file \"$ARGV[1]\"";
76.    while($line1 = <FIRSTFILE>)
77.    {
78.        $line2 = <SECONDFILE>;
79.        chomp($line1);  # removes the CR/LineFeed at the end
80.        print "${line1}${line2}";
81.    }
82.    close(FIRSTFILE);
83.    close(SECONDFILE);
```

84. cut.pl
85. - Demonstrates:
    Merging contents of files

```
86.    #program  cut.pl, a tiny program to print only certain columns
   of a file
87.    if( @ARGV < 3)
88.    {
89.        print "usage: cut.pl filename StartColumn Length";    exit
   0;
90.    }
91.    $StartColumn = $ARGV[1];  #needs more error checking here...
92.    $Length = $ARGV[2];
93.    open(MYFILE,$ARGV[0]) || die "Cannot open file \"$ARGV[0]\"";
94.    while($line = <MYFILE>)
95.    {
96.        print substr($line,$StartColumn-1,$Length)."\n";
97.    }
```

98. List of variable type prefixes

Variable Type	Character Prefix
Scaler	$
Array	@
Hash	%
Subroutine	&

99. List of special characters

Character	Meaning
\n	newline
\"	double quote
\r	carriage Return
\t	tab

100. the substitution operator

The 'g' means 'global', all instances on the line

```
s/OldString/NewString/g
```

or to do case insensitive use the 'i' modifier:

```
s/OldString/NewString/ig
```

101. ReplaceString.pl using the substitution operator, s/OldString/NewString/g

```
#program  ReplaceString.pl
if( @ARGV < 2)
{
    print "usage: ReplaceString.pl filename OldString NewString\n";
    print "   example: perl ReplaceString.pl intelliquest.txt ";
    print "IntelliQuest Kantar > kantar.txt\n";
    exit 0;
}
$OldString = $ARGV[1];
$NewString = $ARGV[2];
open(MYFILE,$ARGV[0]) || die "Cannot open file \"$ARGV[0]\"";
while($line = &lt;MYFILE&gt;)
{
    $line =~ s/$OldString/$NewString/g; # the "g" means "global", all
                                        #instances on the line
    print STDOUT $line;
}
```

Replace all strings in a file from the command line

Just remember p i e

```
perl -p -i -e 's/oldstring/newstring/g' textfile.txt
```

Search.pl

\- Demonstrates:
the matching operator, =~
the index() function to find substrings in strings

```
#program  Search.pl, example of the matching operator, =~
if( @ARGV < 2)
{
    print "usage: Search.pl filename SearchString\n";
    print "   example: Search.pl iq.txt the \n";
    exit 0;
}
$SearchString = $ARGV[1];  #needs more error checking here...
open(MYFILE,$ARGV[0]) || die "Cannot open file \"$ARGV[0]\"";
$i=1;
while($line = <MYFILE>)
{
    if( $line =~ /$SearchString/i ) #does $line contain $SearchString?
                                    # /i means to ignore case
    {
  print STDOUT "$i: $line";
        $StartColumn = index($line,$SearchString)+1;
  print "  (\"$SearchString\" starts in column $StartColumn)\n";
    }
    $i++;
}
```

ColumnSearch.pl

\- Demonstrates:

```
#program  ColumnSearch.pl, shows a string only if in certain columns
if( @ARGV < 4)
{
```

```
148.      print "usage: ColumnSearch.pl filename SearchString
   StartColumn EndColumn\n";
149.      print "   example: ColumnSearch.pl sample.pst 6 20 20\n";
150.      exit 0;
151.  }
152.  $SearchString = $ARGV[1];
153.  $StartColumn = $ARGV[2];
154.  $EndColumn = $ARGV[3];
155.  $Length =  $EndColumn - $StartColumn + 1;
156.  open(MYFILE,$ARGV[0]) || die "Cannot open file \"$ARGV[0]\"";
157.  $i=1;
158.  while($line = <MYFILE>)
159.  {
160.      if( substr($line,$StartColumn-1,$Length) =~ /$SearchString/
   )
161.      {
162.    print STDOUT "$i: $line";
163.      }
164.      $i++;
165.  }
```

166. DBConnection.pl
167. - Demonstrates:
using the "|" operator to pipe output from a system command to a "psuedo-file".

```
168.  #DBConnection.pl
169.  # reads a database using isql on NT with SQLServer and returns
   info from pubs
170.  # 1999/08/21, mitch fincher
171.   MAIN:
172.   {
173.  $SelectStatement = "SELECT au_lname,au_fname,phone FROM authors
   ";
174.  $DBInfo =  "-H mfincher -S mfincher -d pubs";
175.  $SQLCommand = "isql -Uguest -P\"\" $DBInfo -h-1 -s! -w5000 -Q\"
   set nocount on $SelectStatement \"";
176.  print $SQLCommand;
177.  open(RESULTSET, $SQLCommand ." | ") || die "cannot do a sql
   statement";
178.   $i=1;
179.       while($line = <RESULTSET>)
180.       {
181.     chomp($line);
182.     $i++;
183.     ($au_lname,$au_fname,$phone) = split (/\!/,$line);
184.     $au_fname =~ s/ //g;
185.     $au_lname =~ s/ //g;
186.     $phone =~ s/[ ]*$//g; # remove only trailing blanks
187.
188.     print "$au_fname $au_lname\'s phone number is $phone\n";
189.       }
190.      close(RESULTSET);
191.  }
192.
```

5. File Handling
    1. open a file for writing:
```
2.    open(OUTFILE, ">temp.out") || die "Cannot find temp.out";
3.    print OUTFILE "the time has come, the walrus said,...\n";
4.    close OUTFILE;
```
    5. open a file for appending:
```
6. open(APPENDFILE, ">> $filename")
7.                   or die "Couldn't open $filename for appending:
   $!";
```
    8. open a file for reading:

```
open(INFILE, "< $filename")
                    or die "Couldn't open $filename for reading:
   $!";
```

11. read a whole file into an array and remove the linefeed (v5.0):

```
   chomp(@records = <FH>);
```

13. File test operators

Operator	Meaning
-e $file	File exists
-r $file	File is readable
-w $file	File is writable
-d $file	File is a directory
-f $file	File is a file

14. Example:

```
        if( -f "c:/autoexec.bat")
        {
        print "file exists\n";
        }
```

6. Control Structures:

A. for loop

```
      for ($i=1; $i<10;$i++)
         { print "$i\n"; }
```

D. While loop

```
       $j=1;
       while($j < 10)
          {
          print "$j ";
          $j++;
          }
```

K. If-elsif-else

```
$i=0;
if ( $i eq 0 )
      { print "$i = 0"; }
elsif ( $i eq 2 )
      { print "$i = 2"; }
else
      { print "$i != 2, $i !=0"; }
```

T. foreach loop

```
     @names = ("Sarah", "Pam", "Mitch");
     foreach $name (@names)
        { print "$name, "; }
```

2. Array Functions

1. push - pushes an element onto the end of an array
   push (@myarray,$lastvalue) is the same as @myarray = (@myarray,$lastvalue)
2. pop - pops the last value from an array. $value = pop(@myarray)
3. unshift - pushes an element onto the front of an array
4. shift - pops an element from the front of an array
5. reverse - reverses the elements of an array
6. sort - sorts array elements based on ascii values
7. chomp - removes last record separator from each element

2. Assign initial values to an array:

```
     @months = ("January","February","March","April","May","June",
          "July","August","September","October","November","December");
or sans the quotes,
     @months = qw(January February March April May June
```

```
7.               July August September October November December);
8.
9. to use a specific element prepend it with a "$", like $months[$i].
10.
11.
12.
13. misc
14.       $dir = `dir *.pl`;  print $dir  // backtick operator
15.
16.  perl -e 'srand; print int(rand(100))'  //kinda random generator
17.  perl -pi.bak -e "s/OLDSTRING/NEWSTRING/g" filelist  // substitues
   string in files.
18.
19.  # to print a header to a web browser
20.     print "Content-type: text/html\n\n
21.  print STDOUT "Enter some text:"
22.  $mytext = <STDIN>;
23.  rename oldfilename newfilename
24.       renames file, 1 for success, 0 otherwise
25.
26.  to get file info:
27.  stat filehandle
28.  example:
29.  @filestuff = stat "C:/autoexec.bat";
30.  print "filesize=".@filestuff[7];
31.  # how to show the current date?
32.  ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
   localtime(time);
33.  ++$mon; # that wierd month start at zero thingy
34.  print "$mon / $mday / $year";
35.  or if you want the textual month,
36.       ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
   localtime(time);
37.       @months = ("January","February","March","April","May","June",
38.                 "July","August","September","October","November","December
   ");
39.       $myDate = "@{months[${mon}]} ${mday}, ${year}";
40.
41.  to access command line arguments:
42.  $ARGV[0] is the first arg (it skips the program name)
43.  $ARGV[1] is the second arg
44.  $#ARGV is the count of arguments minus 1
45.
46.  Check for an argument
47.  if(@ARGV < 3)
48.  {
49.       print "usage: programname yada yada yada";
50.       exit 0;
51.  }
52.
53.
54.  to create an array from parsing a string for separator tokens use
   "split"
55.  Example to parse the first input string for ~:
56.       @titleArray = split /~/,$ARGV[0];
57.
58.  to get the length of an array:
59.       $len = scalar(@titleArray);
60.  or more simply,
61.       $len = @titleArray;
62.
63.
64.  to convert a string to lower case
65.
66.  mystring =~ tr/A-Z/a-z/; #kinda like the unix tr command
```

```
67.
68.  boolean tests:
69.       == != > <   : tests for numerics
70.       eq ne lt gt : string
71.
72.
73.
74.  to create a new directory
75.   mkdir("mydir", 0755) || die "Cannot create directory mydir: $!";
76.
77.  print <<'ENDOFTEXT';
78.  any sort
79.  of text typed
80.  here will be printed.
81.  ENDOFTEXT
```

82. To configure PERL script mapping on web servers for windows 98

    From some obscure page in the microsoft help database:

    1. Start Regedt32.exe and open HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\ScriptMap
    2. Click Add Value from the Edit menu.
    3. The Value Name is .pl
    4. The Data type is REG_SZ.
    5. The String value is <the full path to perl.exe>\perl.exe %s %s
    6. Restart the WWW service.

83. Subroutine to print a file to stdout

```
84.  sub print_file
85.  {
86.      local ($myfile) = @_;
87.
88.      open(WORDSLIST, $myfile);
89.    while ($name = <WORDSLIST>) {
90.        print "$name";
91.    }
92.    close(WORDSLIST);
93.  }
```

94. Invoke a perl script from a perl script on windows.

    Remember to put the complete path to perl.exe in front of your perl script.

```
    result = system("C:\\Perl\\bin\\perl.exe
ReplaceString.pl \"USESAMPLEMGMT=1\" \"USESAMPLEMGMT=0\" $surveyName\\
$surveyName.ini");
    result /= 256;
```

95. Code fragment to change eol char to '<th>', read a line and put the characters between <th> into an array. This assumes one line in the input file.

```
96.      $/ = '<th>';
97.      $i = 0;
98.      while(<HEADERFILE>)
99.      {
100.   ~s/\<\/?t.\>//g;
101.   print $_."\n";
102.   if ($_ ne "")
103.       {
104.     @fields[$i] = $_ ;
105.     ++$i;
```

```
      }
    }
    close(HEADERFILE);
    $/ = "\n";
</pre>
```

CGI.pl

```
#!/usr/bin/perl
    use CGI qw(:standard);
MAIN:
{
    $query = new CGI;
    $| = 1;
    print $query->header('text/plain');
    $Name = $query->param('Name');

    print "Hello $Name";
}
```

Print the arguments passed in using cgi-lib

```
#!/usr/bin/perl
# this program just prints the name-value pairs passed to it from a
  form
# mitch fincher 2000/07
MAIN:
{
     require "cgi-lib.pl" || die "Cannot find cgi-lib.pl: $!";
     local(*in,$message,$command);
     &ReadParse(*in);
     print &PrintHeader;

     print
  "&lt;html&gt;\n&lt;head&gt;&lt;title&gt;Results&lt;/title&gt;&lt;/head&g
  t;\n";
     print "&lt;body&gt;\n";
     print "&lt;h1&gt;Submission From SimpleForm&lt;/h1&gt;\n";
     print "&lt;h2&gt;Variables passed in:&lt;/h2&gt;\n";

    while ( ($key, $value) = each %in) {
        print "$key = $value&lt;br &gt;\n";
    }
     print "&lt;/body&gt;&lt;/html&gt;";
}
```

Print the environmental variables using cgi-lib

```
#!/usr/bin/perl
# this program just prints the environmental variables available to a
  CGI
# mitch fincher 2000/07
MAIN:
{
     require "cgi-lib.pl" || die "Cannot find cgi-lib.pl: $!";
     local(*in);
     &ReadParse(*in);
     print &PrintHeader;

     print "&lt;html&gt;\n&lt;head&gt;&lt;title&gt;Environmental
  Variables&lt;/title&gt;&lt;/head&gt;\n";
     print "&lt;body&gt;\n";
     print "&lt;h1&gt;Environmental Variables&lt;/h1&gt;\n";
     print "&lt;h2&gt;Variables:&lt;/h2&gt;\n";

```

```
    while ( ($key, $value) = each %ENV) {
        print "$key = $value&lt;br &gt;\n";
    }
     print "&lt;/body&gt;&lt;/html&gt;";
}
```

A simple search engine using cgi-lib

```
#!/usr/bin/perl
# really basic server side search engine for finding one string of text
#   for all the files of a certain extention in one directory.
# This file is a basis for developing a better solution.
# (I'm not proud of this solution, but its a first step).
# For use on windows based system using the "find" command.
# by mitch(at)fincher.org

MAIN:
{
    require "cgi-lib.pl" || die "Cannot find cgi-lib.pl: $!";
    local(*in);
    print &PrintHeader;
    &ReadParse(*in);

    $STRING = $in{searchstring};
# this is the type of files to search
    $FILES2SEARCH = "*.html";
# this is the directory to search
    $DIR2SEARCH = "D:\\inetpub\\users\\mfincher";
    print "Searching Files for \"$STRING\" ...\n&lt;hr /&gt;";
    $command = "find /I \"$STRING\" $DIR2SEARCH\\$FILES2SEARCH";
    open(F, $command ." | ") || die "error";
    $filename = "";
    while (&lt;F&gt;) {
  chomp;
  if( /^---/ ) { # we hit a filename line, they start with "---"
      $firstone = 1;
      $filename = $_;
      $filename =~ s/.*\\//g;
      $filename =~ s/;//g;
  } elsif( length($_) &lt; 1) { # we hit a blank line
  } else {
      if($firstone == 1) { #is this the first from this file?
    print "match in file &lt;a href=\"$filename\"&gt;
  $filename&lt;/a&gt;:";
      }
      print "  &lt;xmp&gt; $_&lt;/xmp&gt;"; #print the line
      $firstone = 0;
  $i++;
  }
    }
    close(F);

    print "\n&lt;hr&gt;search complete.&lt;br&gt;\n";


    if ( $i == 0 )
    { print "Sorry, no matches found for \"$STRING\".\n"; }
    else
    { print "$i matches found for \"$STRING\"\n"; }

}
```

Use atoi to convert ascii strings to integers

```
    $thisyear = ${year}.atoi + 1900;
```

How to read arguments to a subroutine

A special array is created containing all the arguments called "@_". Use $_[n] to access the arguments.

```
CopyFile(thisFile, thatFile);
...
sub CopyFile {
    $inFileName = $_[0];
    $outFileName = $_[1];
    print "\r\n    inFileName = $inFileName";
    print "\r\n    outFileName = $outFileName";
...
}
```

227. Show elapsed time

```
228. my $starttime=time();
229. #do interesting stuff here...
230. printf "\nElapsed Time: %d seconds\n",time()-$starttime;
231.
```

To view the load directories

```
232. perl -e 'print "@INC \n";'
```

233.

"If Python is executable pseudocode, then perl is executable line noise."

"Perl, the only language that looks the same before and after RSA encryption." - Keith Bostic