Applied Perl

Boston University
Information Services & Technology

Course Coordinator: Timothy Kohl

Last Modified: 09/19/09

Outline

- Perl as a command line tool
- Perl in statistics
- Perl in system administration
- Perl and the Web
- Text Processing

Perl as a command line tool.

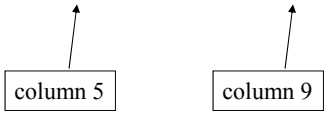
Although the primary mechanism for using Perl is through scripts, Perl can be used on the command line in conjunction with other programs using Unix pipes.

Ex: Take the output of 'ls -als' and print the file names and sizes only.
Typically, the output of ls -als looks like this.

```
4 -rw-rw----   1 tkohl    consrv       310 Sep  7 1999  dead.letter
```

The point being, that (if we number the columns from left to right, starting with **0)** then the two columns of interest are as shown.

```
4 -rw-rw----   1 tkohl    consrv       310 Sep  7 1999  dead.letter
```



The command sequence would be as follows:

```
>ls -als | perl -ane 'print "$F[5] $F[9]\n"'
```

How does this work?

```
>ls -als | perl -ane 'print "$F[5] $F[9]\n"'
```

-e execute the code in quotes
-n execute the code for every line of standard input
(i.e. assume a while(<STDIN>) loop has been wrapped around the code to execute, with each line assigned to **$_**)
-a take the line of standard input and let
@F=split(/\s+/,$_)

The effect is that the output of

ls -als

is split into columns, and then we print out the columns of interest (5 and 9)

Perl's regular expression matching can be put to use on the command line.

Ex: Your Unix path is given by the environmental variable **$PATH**

```
>echo $PATH
```

```
.:/home/tkohl/bin:/usr/vendor/bin:/usr/local/4bin:
/usr/local/bin:/usr/ucb:/usr/bin:/usr/bin/X11
```

If you want a more readable list, you can do the following:

```
>echo $PATH | perl -ne 's/:/\n/g;print'
```

take the path separated by **:**

replace **every** occurrence of **:** with a newline **\n (note we are acting on the variable $_)**

print the result

The result then is

```
.
/home/tkohl/bin
/usr/vendor/bin
/usr/local/4bin
/usr/local/bin
/usr/ucb
/usr/bin
/usr/bin/X11
```

We can even shorten this by using the **-p** option which automatically prints the variable **$_**

```
>echo $PATH | perl -pne 's/:/\n/g'
```

We can also do in-place modification of a file using Perl on the command line.

Ex: Say we wish to replace every occurrence of the word 'Foo' in the file called **somefile** by the word 'Bar'

```
>perl -p -i.old -e 's/Foo/Bar/g' somefile
```

use the print option to print the contents of $_

-i (**in place operation**) and do the modifications to a file called **somefile.old** and then copy it back to the original **somefile**

-e means execute this code

substitution to apply everywhere (g option)

file to modify

Perl in statistics

In this example, we will consider a basic problem in statistics.

For a list of N data points of the form

(x_1, y_1)
(x_2, y_2)
.
.
.
(x_N, y_N)

statisticians consider whether there is some functional relationship between the x and y values.

The most basic possible relationship would be a linear one.

Ideally, we would like a linear function **`y=a*x+b`** such that for each **`i=1..N`**, one has that

$$y_i = a*x_i + b$$

Now, real life data is seldom so neat, so, barring an exact such relationship for all the data, one instead looks for the line of *best fit*, also called the 'regression line' namely the one which minimizes the '**sum of square errors**' that is:

$$SSE = \sum_{i=1}^{n} (y_i - (a+b*x_i))^2$$

The basic problem is to find the **'a'** and **'b'** which minimize this error. In many statistics books you can find the details for deriving these, but in summary, the formulæ for **'a'** and **'b'** are given as follows:

$$a = \frac{N*(\Sigma x_i*y_i) - (\Sigma x_i)(\Sigma y_i)}{N*(\Sigma x_i^2) - (\Sigma x_i)^2}$$

$$b = \frac{\Sigma y_i - a(\Sigma x_i)}{N}$$

Recall that N is the number of data points.

For our example, we will assume that there is a file called **data.dat** with the following entries (where the first column is $\mathbf{x_i}$ and the second $\mathbf{y_i}$) :

1	5.5
3	7.0
4	9.1
7	6.2
11	8.8
15	9.4

Our script will do several things, read in this data set , compute the least squares line according to the formulæ on the previous slide, then we will take the data from the file as well as the formula for the line and plot both using the GNUPLOT program which is available on most Unix systems.

Here is the script:

```
#!/usr/bin/perl
open(DATA,"data.dat");
while($line=<DATA>){
    ($x,$y)=split(/\s/,$line);
    push(@X,$x);
    push(@Y,$y);
}
close(DATA);

($a,$b)=regression(\@X,\@Y);
print "${a}x+$b\n";
```

We read in the file and store the respective x's and y's in two arrays **@X** and **@Y** and then we compute the regression line by passing references to **@X** and **@Y** to a subroutine called **regression()** which computes **a** and **b**.

```
open(GNUPLOT,"|gnuplot -persist");
print GNUPLOT "set origin 0,0;\n";
print GNUPLOT "set yzeroaxis;\n";
print GNUPLOT "set xzeroaxis;\n";
print GNUPLOT "set xrange [0:10];\n";
print GNUPLOT "set yrange [0:10];\n";
print GNUPLOT "set xlabel \"x\";\n";
print GNUPLOT "set ylabel \"y\";\n";
print GNUPLOT "L(x)=$a*x+$b;\n";
print GNUPLOT "plot \"data.dat\",L(x)
;\n";
close(GNUPLOT);
```

Here we invoke the GNUPLOT program as a process with the **-persist** option present to keep the window open after the plot has been made.

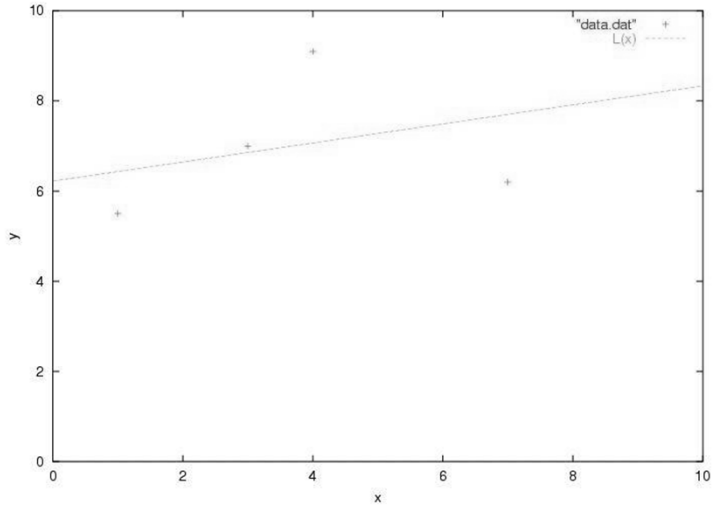
The print lines basically create a GNUPLOT script, the syntax of which can be referenced in the GNUPLOT manual and online.

```
sub regression{
    my @X=@{$_[0]};
    my @Y=@{$_[1]};
    my $N=@X;
    my $i;
    my ($SXY,$SX,$SY,$SX2)=(0,0,0,0);
    my $m,$b;
    for($i=0;$i<$N;$i++){
        $SX+=$X[$i];
        $SX2+=$X[$i]**2;
        $SY+=$Y[$i];
        $SXY+=$X[$i]*$Y[$i];
    }
    $a=($N*($SXY)-($SX)*($SY))/($N*$SX2-$SX**2);
    $b=($SY-$a*($SX))/$N;
    return($a,$b);
}
```

This computes the **a** and **b** of the regression line.

In particular, note that the two parameters are references to the arrays of x and y data which must be dereferenced in order to access them separately within the sub.

Observe now the output on the screen that GNUPLOT pops up. The data points and regression line are graphed simultaneously.



Note, if you want a hard copy of this, say a pdf file, one can modify the script as follows:

```
print GNUPLOT "set terminal postscript enhanced color;\n";
print GNUPLOT "set output \"plot.ps\";\n";
print GNUPLOT "set origin 0,0;\n";
print GNUPLOT "set yzeroaxis;\n";
print GNUPLOT "set xzeroaxis;\n";
print GNUPLOT "set xrange [0:10];\n";
print GNUPLOT "set yrange [0:10];\n";
print GNUPLOT "set xlabel \"x\";\n";
print GNUPLOT "set ylabel \"y\";\n";
print GNUPLOT "L(x)=$a*x+$b;\n";
print GNUPLOT "plot \"data.dat\",L(x) ;\n";
close(GNUPLOT);
`ps2pdf plot.ps`;
```

The first two lines modify the output so that it goes to a postscript file called **`plot.ps`** and the **`ps2pdf`** command converts **`plot.ps`** to pdf format.

Now there are many mathematical and statistical applications that can be handled in Perl as well as many mathematical modules that one can download from CPAN.

Also, there are modules such as GD for graphics applications.

We used GNUPLOT here as it is a generic package that is available on most Unix systems and can be installed in Windows too.

Perl as a system administrator's tool.

In this section we examine Perl's role in system administration.

As many of the files that control the behavior of a Unix system are text files, and since Perl excels at text file processing it is a natural choice for system administrators.

There is also the fact that it takes less time to assemble a Perl script to do a certain task than, say, a corresponding C program to do the same thing.

Problem: To lock the accounts of users who have not logged in within the last 6 months.

Tactic: Check the age (access time) of the .login file in each users home directory.

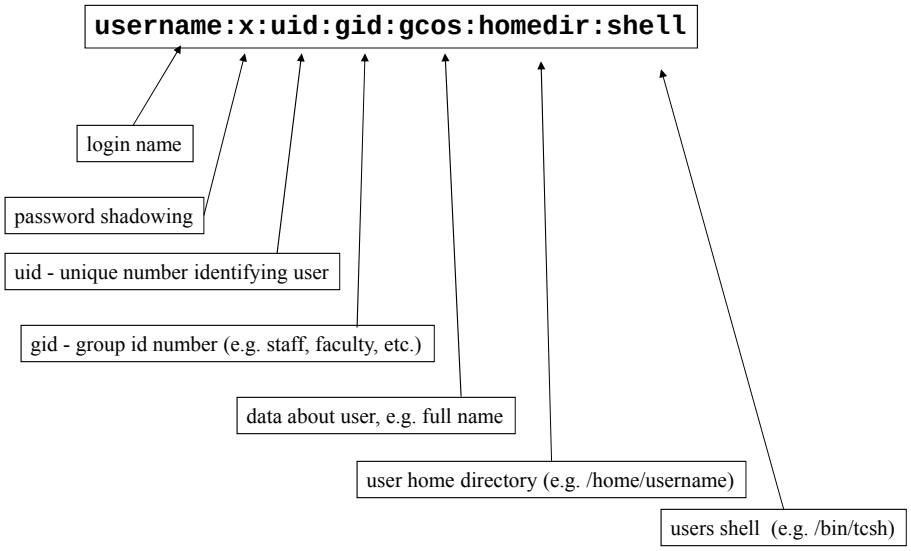
First, how do we get a list of all the 'ordinary' users on the system. Usually, there are two files of importance,

/etc/passwd contains user information

and

/etc/shadow contains encrypted passwords (not needed)

The structure of /etc/passwd looks like this.



Ex:

```
fred:x:3216:25000:Fred Flintstone:/home/fred:/bin/bash
```

3216 is fred's uid and 25000 is his gid.

As such, there may be others with the same gid (i.e. belong to the same group) but only one with that uid.

Since we are interested in looking at the accounts of ordinary users which have only certain types of uids and gids we can, for example, restrict our attention to those in the password file with certain gids

Ex:

25000	- students
25001	- faculty
25002	- staff

For users with one of these gid's we will check to see if they logged in sometime the last 6 months and, if not, lock their account.

So we need to parse the /etc/passwd file and grab the entries with those gid's of interest.

Ex: Let's first look at the password file and print out those lines with one of the gid's we're looking for.

```
username:x:uid:gid:gcos:homedir:shell
```

```
#!/usr/local/bin/perl5
@GID=("25000","25001","25002");
open(P,"/etc/passwd");
while($line=<P>){
      chomp($line);
      @fields=split(/:/,$line);
      foreach $gid (@GID){
            if ($fields[3] ==$gid){
                  print "$line\n";
            }
      }
}
close(P);
```

gid's of interest

open password file

split up each line along **:** and assign to array **@fields**

loop over **@GID** and check

close password file

Ok, now what?

Contained in each line is the home directory of the given user, say /home/username

As such, their .login file is

/home/username/.login

To check the access time, of this file, we can use the **-A** file test operator which returns the number of days since the given file (or directory) was accessed.

So we will use a conditional of the form:

```
if(-A "/home/username/.login" >180){
      # lock their account
}
```

So, here is how the final script might go.

```
#!/usr/local/bin/perl5
@GID=("25000","25001","25002");
$noshell="/bin/nosh";   # void shell prevents login
system("cp /etc/passwd /etc/passwd.save"); # safety first!
open(P,"/etc/passwd");
open(NP,">/etc/newpasswd");
while($line=<P>){
       chomp($line);
       @fields=split(/:/,$line);
       foreach $gid (@GID){
              if ($fields[3] ==$gid){
                     $homedir=$fields[5];
                     if(-A "$homedir/.login" > 180){
                            $line=~s/$fields[6]/$noshell/;
                     }
              }
       }
       print NP "$line\n";
}
close(P);
close(NP);
system("rm /etc/passwd;mv /etc/newpasswd /etc/passwd");
```

Let's break this down.

```
#!/usr/local/bin/perl5
@GID=("25000","25001","25002");
$noshell="/bin/nosh";
system("cp /etc/passwd /etc/passwd.save");
open(P,"/etc/passwd");
open(NP,">/etc/newpasswd");
```

set up gid array

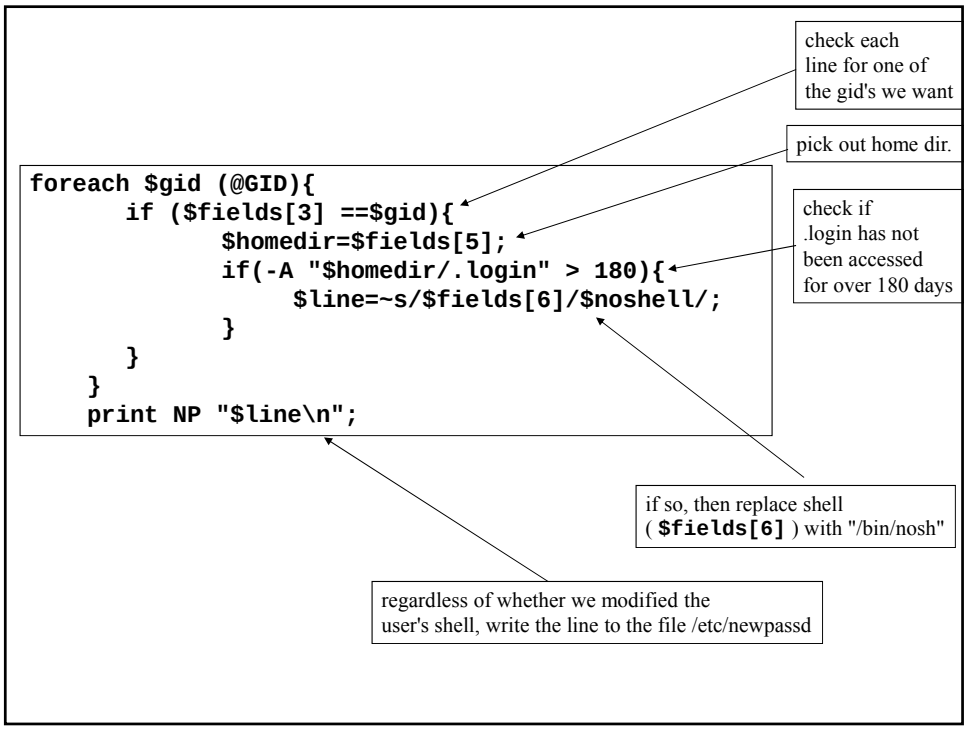
setting a user's shell to /bin/nosh makes logins impossible

make a backup of /etc/passwd

this is the modified version of /etc/passwd

```
while($line=<P>){
       chomp($line);
       @fields=split(/:/,$line);
```

Read in /etc/passwd one line at time and split the fields up along **:**




```
foreach $gid (@GID){
      if ($fields[3] ==$gid){
            $homedir=$fields[5];
            if(-A "$homedir/.login" > 180){
                 $line=~s/$fields[6]/$noshell/;
            }
      }
   }
   print NP "$line\n";
```

```
}
close(P);
close(NP);
system("rm /etc/passwd;mv /etc/newpasswd /etc/passwd");
```

Once done, close both /etc/passwd, and /etc/newpasswd

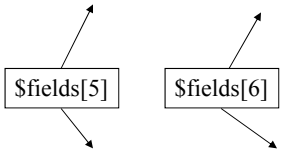
Then remove the old /etc/passwd and replace it with the modified version.

Note, we made a backup of /etc/passwd beforehand in case something went wrong while this script was running.

To clarify, if /home/fred/.login has not been accessed for more than 6 months then this what happens to his entry in /etc/passwd

```
fred:x:3216:25000:Fred Flintstone:/home/fred:/bin/bash
```

becomes




```
fred:x:3216:25000:Fred Flintstone:/home/fred:/bin/nosh
```

Perl and the Web

Perl is used in many ways for web applications, including the management of web servers as well as CGI scripting and more.

Our first example will involve the analysis of web server logs.

In particular we will show how to parse the log files and retrieve the important statistical information contained therein, such as the addresses of those sites connecting to the server as well as content downloaded etc.

This is not strictly speaking a web-centric demonstration, since it will be more about crafting regular expressions to analyze text data, nonetheless it's as good an example of this as any other so...

The basic information that is recorded in any web 'event' which a server might record are:

- the address of the incoming connection (i.e. who visited)
- the time of the connection
- what content they downloaded

Additionally, one may record other data such as:

- any site they came to yours by via a link
- the hardware/software combination they use (e.g. Unix, Windows, Netscape, IE)

Ex: A typical entry in an access_log file:

```
168.122.230.172 - - [16/Feb/2001:08:42:52 -0500] "GET /people/tkohl/teaching/sprin
g2001/secant.pdf HTTP/1.1" 200 0 "http://math.bu.edu/people/tkohl/teaching/spri
ng2001/MA121.html" "Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)"
```

`168.122.230.172`	IP address of visitor
`[16/Feb/2001:08:42:52 -0500]`	time
`"GET /people/tkohl/teaching/spring2001/secant.pdf HTTP/1.1"`	content they retrieved
`200 0`	server response code
`"http://math.bu.edu/people/tkohl/teaching/spring2001/MA121.html"`	referrer
`"Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)"`	client software and archictecutre

```
168.122.230.172 - - [16/Feb/2001:08:42:52 -0500] "GET /people/tkohl/teaching/sprin
g2001/secant.pdf HTTP/1.1" 200 0 "http://math.bu.edu/people/tkohl/teaching/spri
ng2001/MA121.html" "Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)"
```

In order to parse this file and extract the relevant information, say for some statistical analysis or whatever, we need to describe log entries with a regular expression and extract the different components.

Here is a subroutine for parsing entries such as the one above.

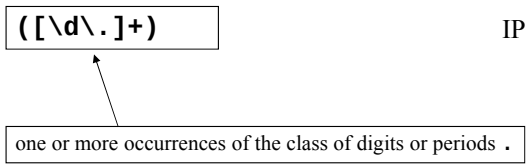
```
sub parse_log{
       my $entry = $_[0];
       $entry =~ /([\d\.]+) \- \- (\[[^\]]+\]) \"([^\"]+)\" (\d+ \d+)
\"([^\"]+)\" \"([^\"]+)\"/;
       return ($1,$2,$3,$4,$5,$6);
}
```

Let's examine the pattern to clarify what's going on.

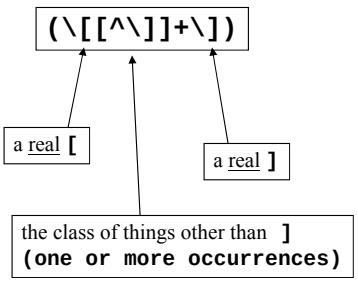
```
168.122.230.172 - - [16/Feb/2001:08:42:52 -0500] "GET /people/tkohl/teaching/sprin
g2001/secant.pdf HTTP/1.1" 200 0 "http://math.bu.edu/people/tkohl/teaching/spri
ng2001/MA121.html" "Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)"
```

Discounting the spaces and dashes between the entries, here are the patterns describing the portions to memorize.

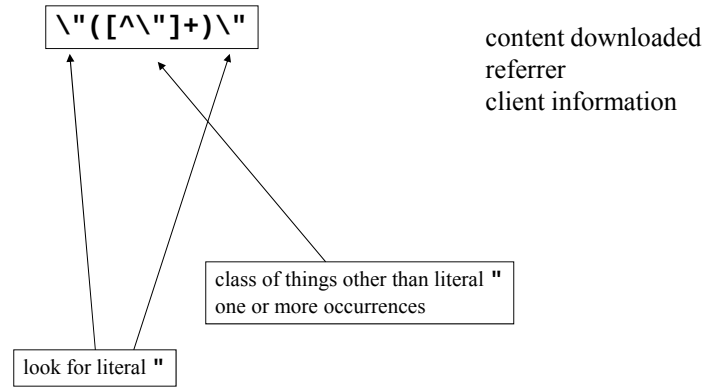
`([\d\.]+)`	**ip address**
`(\[[^\]]+\])`	**date (including the brackets**
`\"([^\"]+)\"`	**content downloaded**
`(\d+ \d+)`	**status code**
`\"([^\"]+)\"`	**referrer**
`\"([^\"]+)\"/`	**client info**



IP address



date



So now, the components of the log entry are returned as an array from the parse_log function.

So we might use it in a larger script as follows:

```
open(LOG,"/usr/local/apache/logs/access_log");
while($line=<LOG>){
       ($ip,$date,$content,$status,$referrer,$client)=parse_log($line);
       # do something with the components
}
close(LOG);
```

Simple Web Clients

Say one wishes to, without using a browser, download some data from a website.

Ex:

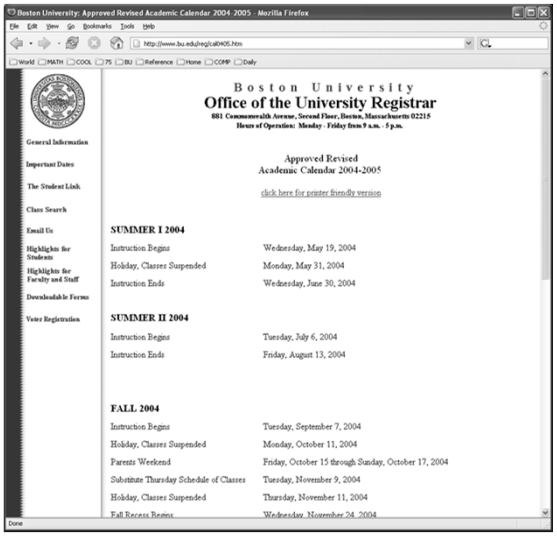
```
#!/usr/bin/perl
use LWP::Simple;
print get($ARGV[0]);
```

call this 'geturl'

```
>geturl http://www.bu.edu
```

The output will be the literal HTML code of the BU homepage, which may not be terribly interesting, but there are other ways of using such data.

Let's consider a more interesting example.
If we wish to find the academic calendar for the 2004/5 academic year, it is located at *http://www.bu.edu/reg/cal0405.htm*



Now suppose we wish to extract the information from this page. The raw output of our script includes a lot of HTML code which certainly isn't essential information.

However, we can extract the information we want by observing that the relevant information we want lies within tags such as these

```
<TD><FONT face="Times New Roman">Instruction Begins </font></TD>
```

what we're after

So we can modify our script, to, in fact, retrieve this URL and then do some custom filtering of the data.

```
#!/usr/bin/perl
use LWP::Simple;
$URL="http://www.bu.edu/reg/cal0405.htm";
@DATA=split(/\n/,get($URL));
foreach (@DATA){
    if(/\<TD\>\<FONT face=\"Times New Roman\"\>(.*)\<\/font\>/){
        $item=$1;
        print "$item\n";
    }
}
```

what we want

which, when run yields

```
Instruction Begins
Wednesday, May 19, 2004
Holiday, Classes Suspended
Monday, May 31, 2004
Instruction Ends
Wednesday, June 30, 2004
Instruction Begins
Tuesday, July 6, 2004
Instruction Ends
Friday, August 13, 2004
.
. etc
```

Let's add a line between each logical entry.

```
#!/usr/bin/perl
use LWP::Simple;
$URL="http://www.bu.edu/reg/cal0405.htm";
@DATA=split(/\n/,get($URL));
foreach (@DATA){
    if(/\<TD\>\<FONT face=\"Times New Roman\"\>(.*)\<\/font\>/){
        $item=$1;
        print "$item\n";
        ($item=~/200(4|5)/) && (print "\n");
    }
}
```

issue a newline if the item ends in **2004** or **2005**

And now the output looks a bit neater:

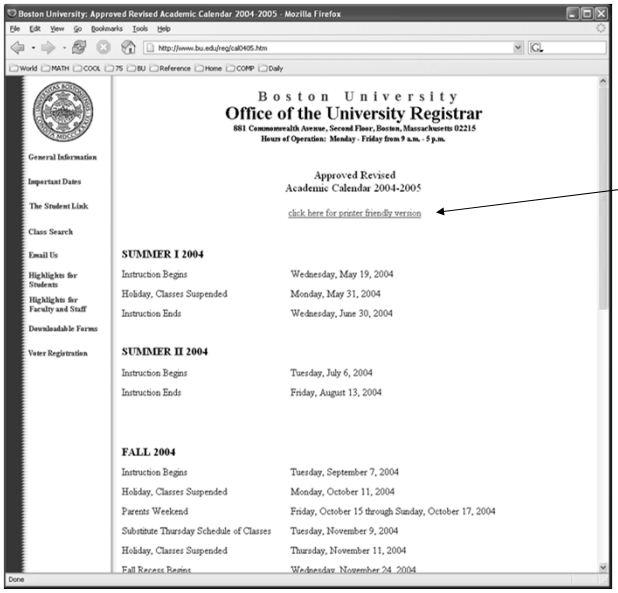
```
Instruction Begins
Wednesday, May 19, 2004

Holiday, Classes Suspended
Monday, May 31, 2004

Instruction Ends
Wednesday, June 30, 2004

.. Etc.
```

Of course, we *could* look closer at the original web page and observe that there is a link to a PDF version of the calendar!



Perhaps we could grab just this file and put it in our home directory.

Indeed, we can!
We note that this link point to the file/URL

http://www.bu.edu/reg/images/cal0405.pdf

So….

```
geturl http://www.bu.edu/reg/images/cal0405.pdf > cal0405.pdf
```

where the '**>**' indicates we should output the result to a file in our home directory also called **`cal0405.pdf`**

We can then view this page at our convenience as follows:

```
acroread cal0405.pdf
```

The point in both cases is that these tools can give one the power to extract data (potentially very volatile data) from a remote site and use it in our own scripts, perhaps with a bit of filtering on our part, but this is easy when using Perl!

Text Processing

In this example, we will analyze the text in a small book and create an index of the words in the book and how often they occur.

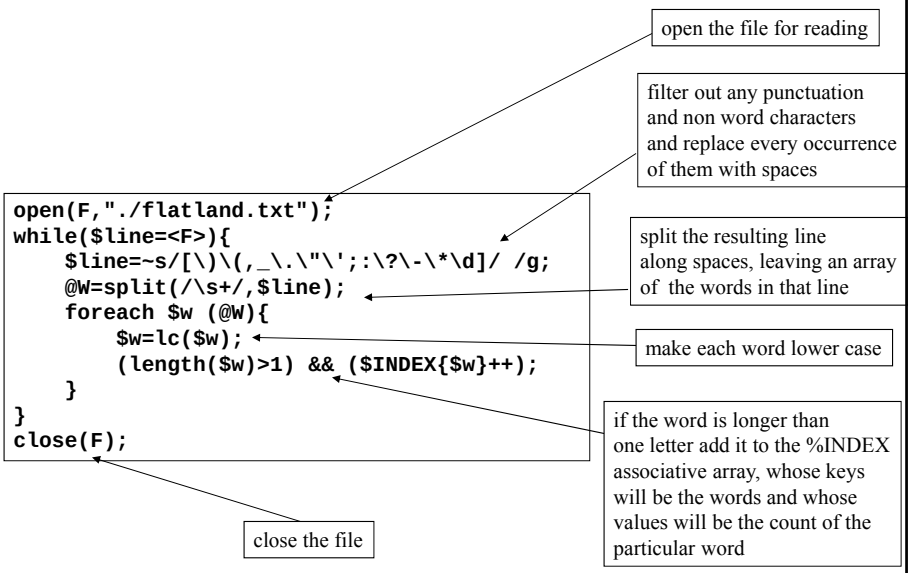
The first part will be to actually obtain a small text to analyze.

```
#!/usr/bin/perl
use LWP::Simple;
$URL="ftp://nic.funet.fi/pub/doc/literary/etext/flatland.txt.gz";
open(F,">./flatland.txt.gz");
print F get($URL);
close(F);
(!(-e "./flatland.txt")) && system("gunzip ./flatland.txt.gz");
```

We use the LWP module to retrieve the compressed text of the book Flatland which we download to the current directory and then uncompress using the '**`gunzip`**' command for uncompressing **`.gz`** files.

On a Windows system, you can just download the file and uncompress it manually.

Next comes the actual reading and indexing of the words in the text.



Now, we need to organize this information to see what are the most common words in the text. In particular, we wish to sort the list according to the size of the word counts.

First, we should demonstrate how one sorts an array of numbers by their numerical value.

Recall that there is a built in **`sort()`** function but that this sorts based on the *dictionary* ordering of the array elements which can lead to unexpected results

Ex:

```
@X=(222,1,10,11,10);
@X=sort(@X);
print "@X";
```

yields

```
1 10 101 11 222
```

To sort by numerical ordering, we use the following technique, which basically manipulates the criterion used to compare elements of the array.

```
@X=(222,1,10,11,10);
@X=sort bynum (@X);
print "@X";

sub bynum{
 $a <=> $b;
}
```

yields

```
1 10 11 101 222
```

bynum is a subroutine which controls the comparison criterion for sort

`$a` and **`$b`** are two elements being compared and **`<=>`** (the spaceship operator!) basically returns -1, 0, or 1 depending on the value of **`$a-$b`**

Now, this technique can be extended to sort the keys of the **`%INDEX`** hash to order it based on the size of the word counts.

```
@WORDS=sort( bycount (keys(%INDEX)));
@WORDS=reverse(@WORDS);

for($i=0;$i<=19;$i++){
    print "$WORDS[$i] -> $INDEX{$WORDS[$i]}\n";
}

sub bycount{
    $INDEX{$a} <=> $INDEX{$b};
}
```

here we sort the keys (words) in **%INDEX** according to the *value* associated to each word, namely the count

then we reverse the array since we wish to see the top 20 words

Lastly, the for loop simply prints out the 'Top 20' words by their count in the text.

```
the -> 2083
of -> 1482
and -> 1022
to -> 1008
in -> 639
that -> 477
is -> 396
you -> 348
my -> 319
it -> 312
as -> 311
by -> 300
not -> 296
but -> 271
for -> 237
be -> 232
with -> 225
or -> 219
at -> 185
his -> 181
```

These results aren't terribly surprising, but this program can be easily modified to do many other similar analyses.

The possibilities are endless.

References for further information on Perl

Books

- Learning Perl by Randal L. Schwartz & Tom Christiansen (O'Reilly)
- Algorithms with Perl by J. Orwant, J. Hietaniemi, J. Macdonald (O'Reilly)
- Programming Perl by Larry Wall, Tom Christiansen and Jon Orwant (O' Reilly)
- Perl Cookbook Tom Christiansen and Nathan Torkington (O' Reilly)
- Web Client Programming in Perl by Clinton Wong (O' Reilly)
- Perl for System Administration by David N. Blank-Edelman (O' Reilly)

Web

http://www.perl.com

http://www.perlmonks.org

http://www.cpan.org

http://math.bu.edu/people/tkohl/perl ← My Perl Page!

Applied Perl

Boston University
Information Services & Technology

Course Coordinator: Timothy Kohl