

WWW Access-Statistics with Pgperl

Dominique F. Thiébaud

thiebaut@cs.smith.edu

http://cs.smith.edu/~thiebaut

One of the common reactions one has after setting up a homepage on the *World Wide Web* (WWW) is to find out how many visits it gets from people around the world. In this article I will describe a way to gather the statistics of page accesses over a variable period of time, and how to incorporate the resulting graphs in the same homepage. The processing will be done in a Unix environment, using Pgperl, an implementation of PGPLOT ported to the elegant language Perl.

Tracking WWW access-statistics

When you browse pages on the WWW and load a document from a site, the server at that site automatically records your access in a log file. This log file contains detailed information about accesses to all *gif* and *jpeg*² graphics files, as well as to all html documents kept at that site. This file is in general called `http.log` or `access.log`, in a subdirectory called `logs`. If you can't find it, ask your system administrator for its location. Here is an example of entries in the log:

```
sorrel.hensa.ac.uk - -[15/May/1996:12:33:28 -0400] "GET /~orourke/upleft.gif HTTP/1.0" 200 4517
sorrel.hensa.ac.uk - -[15/May/1996:12:33:33 -0400] "GET /~thiebaut/smithbgd.jpg HTTP/1.0" 200 16408
petrescu.sfos.ro - - [15/May/1996:12:33:38 -0400] "GET /~thiebaut/chap7-5.html HTTP/1.0" 200 26734
```

The three lines above show three separate accesses. The first two from someone in the U.K., at Site `sorrel.hensa.ac.uk`, and the third one from someone in Romania, at a computer named `Petrescu.sfos.ro`. The first person downloaded two graphic files, `upleft.gif` from User Orourke, and `smithbgd.jpg` from User thiebaut, while the second person requested the html document `chap7-5.html` from User thiebaut.

¹ Dominique Thiébaud is Associate Professor in the Dept. of Computer Science at Smith College, Northampton, MA 01063.

² Gif and Jpeg are two of the most popular formats for encapsulating graphics images for use on the World Wide Web.

The first field of each line is the name of the remote host from which the request originated. The second field is the date and time of the request. The third field, in quotes, is the request line, exactly as it came from the remote browser, followed by a status code returned to the client, and finally the size of the file transferred, in bytes.

Since the typical installation of homepage documents require them to be stored in a public subdirectory in your home directory, your username will always appear in the third field, like `thiebaut` in the last two lines of the example above. This makes it easy to find out statistics about geographical distribution of the source of requests to your homepage, how often

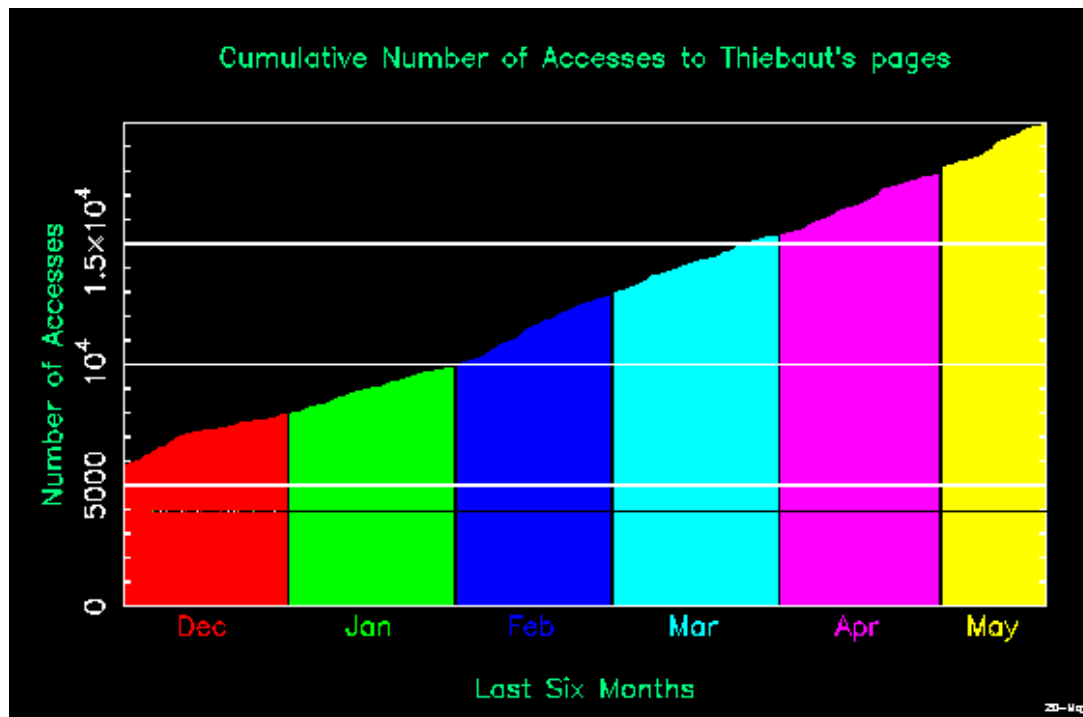


Figure 1: Example of access statistics recorded for a set of homepage.

these requests arrive, and which of your documents are more popular.

Figure 1 shows an example of the type of graph we can generate from the information gleaned from the log file. It is a chart representing the cumulative number of accesses to documents in a homepage over a period of six months.

Let's look at the requirements for generating such a graph, and for incorporating it in your homepage, and adapting it so that it reflects your own statistics.

Requirements

The first requirement for the graph is that it should be generated non-interactively. The second is that its format should be either `gif` or `jpeg` to be compatible with WWW browsers. Finally, the graph should be updated on a regular basis, to reflect up-to-date statistics. The Unix `at` command offers an easy solution to the latter requirement, allowing shell files to run at regular intervals. We will use it to update the graph every 24 hours. The first two requirements, non-interactive generation and gif-format are easily met by using `pgperl`, which I will now explore.

Pgperl

Pgperl is a set of *PGPLOT* Fortran libraries ported to Perl by Karl Glazebrook (see internet contact in internet box below), and should not be confused with the package of the same name which is an interface to the Postgress database.

Perl (see “Perl by Example” by Paul Dubois in *CIP* 7:5, 1993, p. 545) is rapidly growing as a powerful tool, incorporating the power of *shell* files, *awk* and *sed*, two powerful Unix utilities, under one interpreted programming language. `Pgperl` is a module containing *PGPLOT* graphic functions that can be used from within a Perl program. If you have used *PGPLOT* before, then you will find the transition to `Pgperl` very easy. But with `Pgperl`, no need for a fortran compiler! the *PGPLOT* functions are precompiled and can be called directly from within a Perl program. Perl and the `Pgperl` package are all you need to have.

A First Example

Let's start exploring `Pgperl` by generating a graph plotting the following set of (x,y) points:

```
1 300
2 252
3 500
4 452
5 346
6 456
```

You could imagine, for example, that they represent the total number of accesses to some homepage over the past six months. This is a good introduction to the more sophisticated program we'll see next.

We first need to create a file containing these numbers. Let's name it `access.data`. The next step is to write a perl³ program that will read the data file, and generate a graph.

```
#!/local/bin/perl
# example.pg
# Generates a graph from a text file of (x,y) coordinates,
# and stores the graph in a gif file.
#
require "pgplot.pl";           # Load the PGPLOT module

# open data file. Stop if not found.
open(FILE,"access.data") || die "Data file not found\n";

&pgbeg(0,"?",1,1);           # open plot interactively

&pgenv(1,6,0,500,0,0);       # set data limits for axes

&pglabel("Months","Accesses","Statistics of Accesses"); # Labels

while(<FILE>){               # read data file
    ($x[$i], $y[$i]) = split(' '); # split each line into two numbers
    $i++;
}

&pgslw(4);                   # Set line width
&pgline($i,*x,*y);          # Draws a line between the points

&pgend;                       # Close plot
```

Listing 1: example.pg pgperl program

The structure of this perl program is typical of most Pgperl programs, and follows standard steps:

- 1) specify the use of an external graphics library (`require` statement),
- 2) open the graphics display with `pgbeg`,
- 3) define the scale and appearance of the axes with `pgenv`,
- 4) use `pglabel` to optionally define the title and labels associated with the axis,
- 5) format the data to be displayed into array variables,
- 6) optionally define the line style for the graph with `pgslw`,
- 7) draw the line representing the data variation with `pgline`, and

³ I am using Perl 4 in all the example, as Perl 5 may not has been installed on all systems yet.

8) close the graph with `pgend`.

The Perl syntax requires that the graphics functions be prefixed by an `&`-sign. They are fully compatible with the PGPLOT functions described in PGPLOT manuals available on-line from many web sites (see internet box for more information). The main advantage of `pgperl` is that no fortran compilation is required, and the powerful data-processing capability of perl can be integrated seamlessly with the graphics power of PGPLOT.

Because I am running the program interactively at this point, I open the plot using the `&pgbeg` statement with a "?" string. This forces the program to prompt me for the type of output device I want to use, the list of which is shown in Table 1:

GIF	Graphics Interchange Format file, landscape orientation
VGIF	Graphics Interchange Format file, portrait orientation
LATEX	LaTeX picture environment
FILE	PGPLOT graphics metafile
NULL	Null device, no output
PPM	Portable Pixel Map file, landscape orientation
VPPM	Portable Pixel Map file, portrait orientation
PS	PostScript file, landscape orientation
VPS	PostScript file, portrait orientation
CPS	Colour PostScript file, landscape orientation
VCPS	Colour PostScript file, portrait orientation
TEK4010	Tektronix 4010 terminal
GF	GraphOn Tek terminal emulator
RETRO	Retrographics VT640 Tek emulator
GTERM	Color gterm terminal emulator
XTERM	XTERM Tek terminal emulator
ZSTEM	ZSTEM Tek terminal emulator
WD	X Window Dump file, landscape orientation
VWD	X Window Dump file, portrait orientation
XDISP	<code>pgdisp</code> or <code>figdisp</code> server
XWINDOW	X window <code>window@node:display.screen/xw</code>

XSERVE	A /XWINDOW window that persists for re-use
---------------	--

Table 1: Supported output devices for pgperl.

When working at my work station, I use `Xwindow` or `Xterm`. Figure 2 shows the resulting plot once the output device is selected.

Let's now look at the problem of scanning the access log maintained by your local http server, and creating two columns of numbers: one representing days, the second representing the

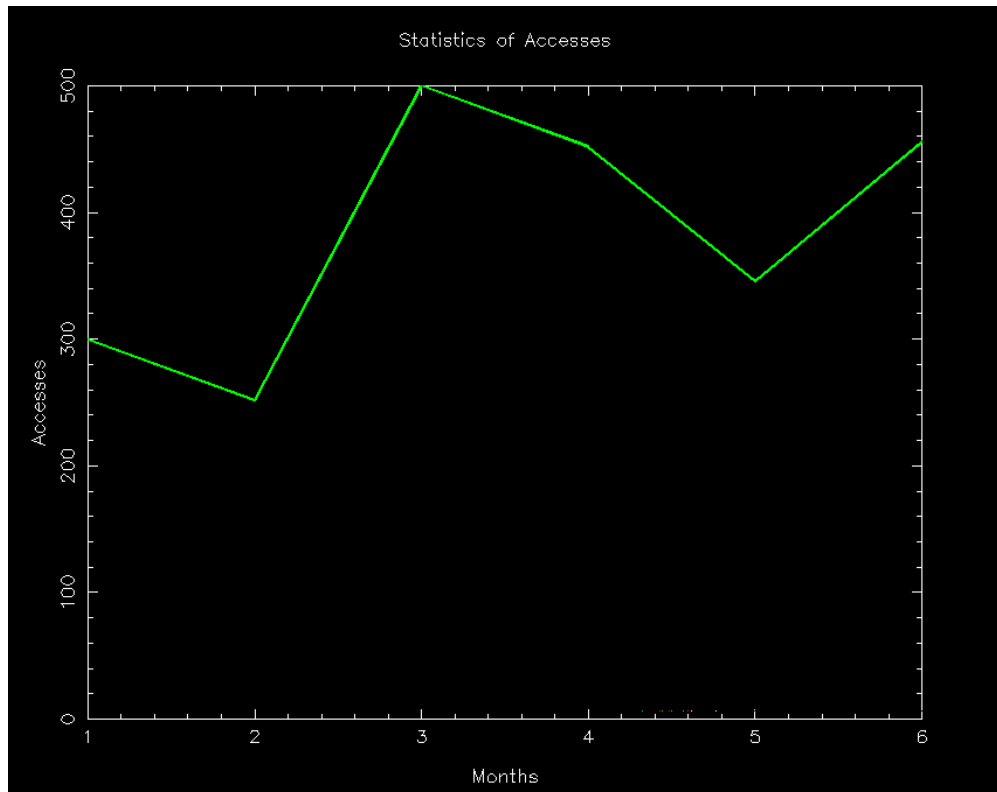


Figure 2: Plot of data used in first example.

number of accesses to your homepage for each day. Keeping programs organized in a modular fashion has great virtues, and I will keep the task of scanning the log file different from that of plotting the result. This yields two programs, one purely perl, and the other one pgperl.

The perl program is shown below. To adapt it to your own environment, you will need to modify the first line and enter the path where your perl interpreter resides, and the first three lines of perl code, initializing the *username*, *logfile*, and *datafile*. For the username, check the log file and find a string that is unique to all accesses to your Web documents.

```

#! /local/bin/perl
# getstats.pl
# Computes cumulative hits to homepage on a day-to-day basis,
# by reading contents of http server log file.
# Output is stored in a file containing two columns, the first one
# counting the days, the second one itemizing the cumulative
# number of accesses over those days.

$username = "thiebaut";      # unique to all accesses to my homepage
$logfile = "/local/WWW/logs/access.log";  # where server keeps log
$datafile = "cummul.accesses.dat";      # where output data are stored

# --- open log file ---
open(LOGFILE, $logfile)
  || die "could not open log file $logfile\n";

# --- scan log until username appears ---
do {
  $_ = <LOGFILE>;
} until /$username/;

# --- get date of first recorded access to my homepage ---
&GetDate($_);
$lastdate = "$day"."$month"."$year";

# --- open output data file ---
open(DATFILE, ">$datafile");

# --- scan log file ---
$NoDays = 1;
$NoAccesses = 0;

while (<LOGFILE>) {
  #--- skip gif and jpeg accesses ---
  next if (index($_, ".gif") != -1);
  next if (index($_, ".jpg") != -1);

  #--- look for accesses that contain username ---
  if (index($_, $username) != -1) {
    $NoAccesses++;
  }

  #--- get date from log line ---
  &GetDate($_);
  $currentdate = "$day"."$month"."$year";

  #--- if we have started a new day, record it ---
  if ($currentdate ne $lastdate) {
    $lastdate = $currentdate;
    print DATFILE "$NoDays $NoAccesses\n";
    $NoDays++;
  }
}

#--- if only entries for one day, catch it now ---
if ($NoDays == 1) {
  print DATFILE "$NoDays $NoAccesses\n";
}

#--- add two more entries to data file to polygone
print DATFILE "$NoDays 0\n";      # bring cummul. to 0
print DATFILE "1 0\n";           # bring line to day 1, 0 accesses
close(DATFILE);

```

```

close(LOGFILE);

# -----
# GETDATE
# Gets a string as parameter and isolates the date
# in it *****[dd/mmm/yy:hh:mm:ss]*****
# Uses $_ but returns it to its original value.
#
# Output returned in: $day, $month, $year
# -----
sub GetDate
{
    local($line) = @_;
    local($dummy);
    local(%TwelveMonth) = ('Jan','01','Feb','02','Mar','03',
                           'Apr','04','May','05','Jun','06',
                           'Jul','07','Aug','08','Sep','09',
                           'Oct','10','Nov','11','Dec','12');

    # --- get date from line of text. format of log line =
    #
    #      computer.domain.edu -- [day/month/year:time] "junk"
    #
    $first = index($line,"\[")+1;
    $last  = index($line,":")-1;
    $dummy = $_;          # save current line
    $_ = substr($_,$first,$last-$first+1);
    /(\d+)\./(\w+)\./(\d+)/;
    $day   = $1;
    $month = $TwelveMonth{$2};
    $year  = $3;
    $_     = $dummy;      # restore current line
}

```

Listing 2: *getstats.pl*.

I will refer you to Paul Dubois' excellent tutorial on Perl in *CIP 7:5* for background information on the perl program in Listing 2. Its companion pppperl program, shown below, takes as input the file generated by the previous perl program, and generates a graphics file, *accesses.gif*, that contains the plot.

```

#!/home/thiebaut/bin/pppperl
# dogif.pg
#
# Syntax: dogif.pg datafile
#
# Takes the file generated by getstats.pl and plot its content to a
# gif file.

require "pgplot.pl";

# --- Global variables. Modify to reflect your settings ---
$giffile = "accesses.gif";
$datafile = "cummul.accesses.dat"; # where output data are stored
$outputdevice = "$giffile/GIF";   # where PGLOT file will be stored

# --- generate graph outline ---

```

```

&pgbeg(0,"$outputdevice",1,1);
                                # open file, with only 1 graph
&pgpap(7,0.618);                # resize output to 7" width, and .618 ratio
&pgscf(1);                      # select character font (1=normal 2=roman)
&pgslw(2);                      # set line width 4 * 0.005 inches
&pgsch(1.6);                   # 1.6 times 1/40th height of view space

# --- determine min and max values of x and y ---
$firstline = `head -1 $datafile`;
($minx,$y) = split(' ', $firstline);
$lastline = `tail -3 $datafile | head -1`;
($maxx,$maxy) = split(' ', $lastline);
&pgenv($minx,$maxx,$miny,$maxy,0,0); # set scale independent of each
                                # other and label coordinates

&pgsci(10);                    # set color of labels to green-cyan
&pglabel("Days",              # label axes and plot
         "Number of Accesses",
         "Cumulative Number of Accesses");

# --- open and read data file ---
open(FILE,$datafile) || die "Could not open $datafile\n";
$i=0;
while (<FILE>) {
    # --- get x y pairs into two arrays
    ($x[$i],$y[$i]) = split(' ');
    $i++;
}
close(FILE);

# --- display curve and fill it with color ---
&pgsci(2);                    # set color index to red
&pgslw(2);                    # increase line width
&pgpoly($i, *x, *y);         # draw it as a closed polygone

# --- plot the frame around the box ---
&pgsci(1);                    # white
&pgbox("BCNSTP",0.0,0,"BCNSTP",0.0,0);
                                # draw labeled frame around viewport

# --- close the graph ---
&pgend                        # close point

```

Listing 3: Dogif.pg.

First the program defines the plot characteristics, how many plots will be stored in the graphic file (just one for our application), the size and aspect ratio of the plot, the font used to draw the characters, the line-width, and how much of the graphic area the plot will cover. The next order of business is to define the scale for the axes, the labels for both axes and the title. Once this is done, the data file is read, and its contents put into two 1-dimensional arrays, which are used by `pgpoly` to draw a closed polygon. Using a closed polygone instead of a regular line permits the area under the curve to be colored. The axes and tick marks are drawn **after** the data so that the tick marks can appear on top of the curve. The `pgbox` call takes several (obscure)

options that are specified by a string of characters (“BCNSTP”). They control the appearance of lines and tick marks on the graph. See the PGPLOT manual for more information.

The result is shown in Figure 3.

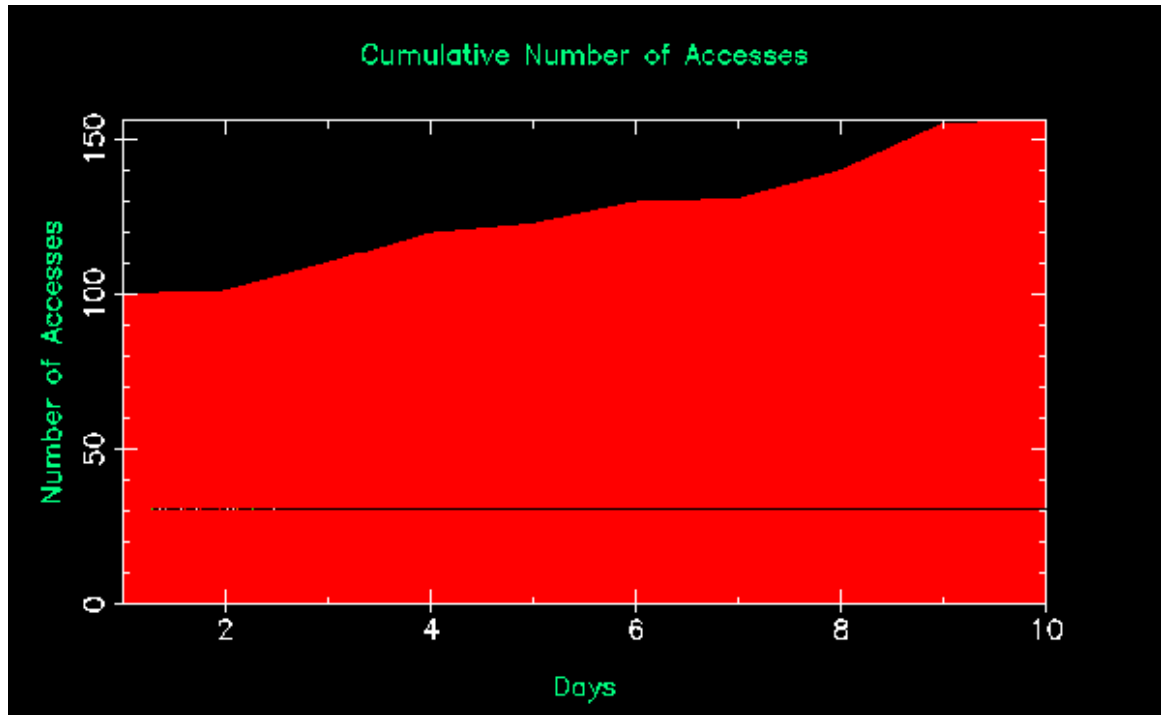


Figure 3: Output of *dogif.pg*.

More Refined Statistics

The method developed above can easily be adapted to generate a plot where the accesses are broken down by month, each month plotted in a different color as shown in Figure 1. To accomplish this, we rely, once again, on two programs. `Getstats.bymonth.pl`, a perl program, is similar to `getstats.pl`, but keeps all the accesses relating to a given month in a separate file. Its output thus spans separate files with names of the form `data.04.1996`, `data.05.1996`, `data.06.1996`, and so on. Its pperl companion, `dogif.bymonth.pg` generates the plot of the accesses as a function of months, and is called with a variable number of files specified on the command line. The command

```
dogif.bymonth.pg data.04.1996 data.05.1996 data.06.1996
```

for example, reads three data files for the April, May and June 1996, and generates the plot for those three months only. For the sake of brevity, I have not included the listings of the two

programs here, but they can easily be retrieved via anonymous ftp for the site indicated in the Internet box at the end of this article.

Gathering Statistics of the Geographical Source of Requests

In the previous sections I described sophisticated graphical tools to generate WWW browser-compatible graphic plots. Sometimes, though, a little ingenuity can allow accurate histograms to be included in a homepage without the use of sophisticated graphics packages. The IMG tag used to include a graphic image in an html document has a width field which allows the user to specify the width of the image, in number of pixels. If the image file contains a simple rectangle filled with a given color, then by specifying different values for the width, we can generate bars of varying lengths. This is exactly what is used to generate Figure 4 which shows a histogram of the number of accesses by country of origin. Generating this histogram is implemented as an html table with two columns. The first one contains the name of the country, and the second one contains the image of a filled rectangle whose width is proportional to the number of accesses. According to the example in Figure 4, 110 accesses to my homepage originated in Argentina, 182 in Australia, etc.

```
<TABLE COLSPEC="R20 L20">
<TR><TH><STRONG> WWW Domain</STRONG> </TH>
  <TH><STRONG> Accesses</STRONG></TH></TR>
<TR><TD>Argentina</TD>
  <TD><IMG SRC="bar.gif" width=110 height=10 align=left>110</TD></TR>
<TR><TD>Australia</TD>
  <TD><IMG SRC="bar.gif" width=187 height=10 align=left>187</TD></TR>
<TR><TD>Austria</TD>
  <TD><IMG SRC="bar.gif" width=14 height=10 align=left> 14</TD></TR>
<TR><TD>Belgium</TD>
  <TD><IMG SRC="bar.gif" width=174 height=10 align=left>174</TD></TR>
</TABLE>
```

Listing 4: html table using variable width fields to generate a histogram.

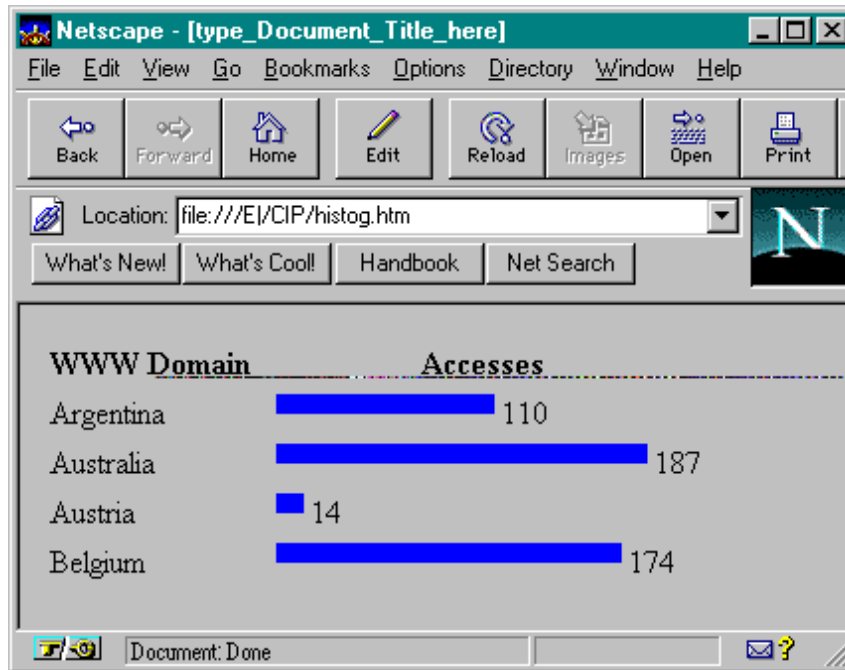


Figure 4: histogram created with an html table tag.

Only four countries are shown, and the actual pixel-width of the bars is shown here as scaled 1:to:1 with the actual number of accesses, which is shown on the right-hand side of the bar. A C program (`histogram.c`) that scans the http server log, and that generates an html table containing such an histogram is available via anonymous ftp. See Internet box at the end of this article.

Updating the Statistics at Regular Intervals

I have covered some of the basics for processing the information stored in the log kept by an http server, and shown how to generate statistics of access using perl and pppperl programs. But these programs will take a long time to execute, given the enormous amount of information that can potentially reside in the log file (the log file for my department, for example, grows at an approximate rate of 800,000 accesses a year, or 80 MBytes/yr). Moreover, the instantaneous nature of the World Wide Web requires those graphic statistics to be updated regularly and frequently. In a Unix environment, a shell file and the `at` utility are a natural choice for this task. A shell file is simply a collection of commands that, instead of being typed at the keyboard by the user, are stored in a text file. When the user wants to execute them, he or she simply types the name of the shell file. The Unix operating system then automatically processes each command in the shell file, exactly as if they had been typed interactively. The `at` command is a

standard Unix command that specifies when a given command should be executed. Its format is very flexible, as the two examples below illustrate:

```
at 815am Jun 6 mail students < grades
at now + 1 day rm exam.sheet
```

The first line mails a file called `grades` to students (an alias for several users) at exactly 8:15 a.m. on June 6. The second example shows how a user can issue a command (deleting the file `exam.sheet`) and make it take effect 24 hours later.

Listing 5 shows a shell file, `dostats`, that uses the programs introduced earlier to automatically process the http log, generate several data files of accesses for each month, create a gif file containing the plot of the accesses over the last six months. The shell file reschedules itself to run the following day, at 4:00 a.m.

```
#!/bin/sh
# dostats
# computes statistics of access to Thiebaut's homepage every day
# and update gif graph of monthly accesses.
# scan log and break accesses by month
getstats.bymonth.pl

# generate the list of files covering the last 6 months
FILES=`ls data.* | sort | tail -6 | awk '{printf ("%s ", $1)}'`

# generate the gif plot of the last 6 months
dogif.bymonth.pg $FILES

# reschedule ourselves for tomorrow
at 4:00am tomorrow /home/thiebaut/bin/dostats > /dev/null 2>&1
```

Listing 5: A self-scheduling shell file generating the plot of Figure 1.

Note the redirections used on the last line. "`> /dev/null`" is the standard way in Unix to say "every thing that might be displayed by the `at` command when it executes should be discarded". The "`2>&1`" expression indicates that if an error message is displayed by the `at`-command, it, too, should be discarded. These two redirection expressions are necessary for commands that may output information to the screen as they run, since the user is not there to see the output of the file.

Conclusions

Creating and maintaining a Web page often require a lot of time and energy, but is a satisfying activity that can benefits many anonymous Internet explorers. Gathering the statistics

of how often your homepage documents are accessed is one way to glean the reward of the hard work and long hours spent in front of the computer. With the elegance and sophistication of the web-oriented language perl and of its ppperl extension, gathering statistics and creating graphics in a non-interactive fashion is simple and easy to setup. Using standard Unix commands, these program can be executed automatically and at regular intervals without user-intervention and automatically update your Web documents.

Internet links and addresses of interest

Information on the WWW links, site and email addresses mentioned in this article.

Karl Glazebrook, (kgb@aaaoepp.aao.gov.au) originated ppperl, and maintains a Web page on ppperl related topics: <http://www.aao.gov.au/local/www/kgb/ppperl.html>

The Ppperl package can be obtained from the following site <http://www.ast.cam.ac.uk/~kgb/ppperlftp.html>

A nice Perl tutorial can be found at <http://agora.leeds.ac.uk/Perl/start.html>

For information on the format of the log file kept by the http server, see <http://www.w3.org/hypertext/WWW/Daemon/User/Config/Logging.html#LogFormat>

The PGPLOT manual can be obtained from <http://www.jach.hawaii.edu/~frossie/user/ppperl/pgplot.html>

The programs and example files used in this article can be downloaded from <http://cs.smith.edu/~thiebaut/ComputersInPhysics>, or via anonymous ftp from [cs.smith.edu](ftp://cs.smith.edu), in the `pub/thiebaut/ComputersInPhysics` directory.