1   Cute Tricks With Perl and Apache

1.1 Description

Perl and Apache play very well together, both for administration and coding. However, adding mod_perl to the mix creates a heaven for an administrator/programmer wanting to do cool things in no time!

This tutorial begins a collection of CGI scripts that illustrate the three basic types of CGI scripting: dynamic documents, document filtering, and URL redirection. It also shows a few tricks that you might not have run into -- or even thought were possible with CGI.

Then, we move to look at different uses of Perl to handle typical administrative tasks. Finally, we continue with the next step beyond CGI scripting: the creation of high performance Apache modules with the mod_perl API.

1.2 Part I: Tricks with CGI.pm

`CGI.pm` is the long-favoured module for CGI scripting, and, as mod_perl can run CGI scripts (mostly) unaltered, also provides significant advantages for mod_perl programmers. Let's look at some of the more interesting uses of this module in web programming.

1.2.1 Dynamic Documents

The most familiar use of CGI is to create documents on the fly. They can be simple documents, or get incredibly baroque. We won't venture much past the early baroque.

1.2.1.1 Making HTML look beautiful

<I> <hate> <HTML> <because> <it's> <ugly> <and> <has> <too> <many> <$#@*&> <angle> <brackets>. With CGI.pm it's almost good to look at. Script I.1.1 shows what a nested list looks like with CGI.pm.

```
Script I.1.1: vegetables1.pl
----------------------------
#!/usr/bin/perl
# Script: vegetables1.pl
use CGI::Pretty ':standard';
print header,
    start_html('Vegetables'),
    h1('Eat Your Vegetables'),
    ol(
        li('peas'),
        li('broccoli'),
        li('cabbage'),
        li('peppers',
            ul(
                li('red'),
                li('yellow'),
                li('green')
              )
          ),
        li('kolrabi'),
```

```
        li('radishes')
        ),
    hr,
    end_html;
```

1.2.1.2 Making HTML concise

But we can do even better than that because CGI.pm lets you collapse repeating tags by passing array references to its functions. Script 1.2 saves some typing, and in so doing, puts off the onset of RSI by months or years!

```
Script I.1.2: vegetables2.pl
----------------------------
#!/usr/bin/perl
# Script: vegetables2.pl
use CGI ':standard';
print header,
    start_html('Vegetables'),
    h1('Eat Your Vegetables'),
    ol(
        li(['peas',
            'broccoli',
            'cabbage',
            'peppers' .
            ul(['red','yellow','green']),
            'kolrabi',
            'radishes'
        ),
    hr,
    end_html;
```

Or how about this one?

```
Script I.1.3: vegetables3.pl
----------------------------
#!/usr/bin/perl

# Script: vegetables3.pl
use CGI::Pretty qw/:standard :html3/;

print header,
    start_html('Vegetables'),
    h1('Vegetables are for the Strong'),
    table({-border=>undef},
        caption(strong('When Should You Eat Your Vegetables?')),
        Tr({-align=>CENTER,-valign=>TOP},
          [
           th(['','Breakfast','Lunch','Dinner']),
           th('Tomatoes').td(['no','yes','yes']),
           th('Broccoli').td(['no','no','yes']),
           th('Onions').td(['yes','yes','yes'])
          ]
        )
        ),
    end_html;
```

1.2.1.3 Making Interactive Forms

Of course you mostly want to use CGI to create interactive forms. No problem! CGI.pm has a full set of functions for both generating the form and reading its contents once submitted. Script I.1.4 creates a row of radio buttons labeled with various colors. When the user selects a button and submits the form, the page redraws itself with the selected background color. Psychedelic!

```
Script I.1.4: customizable.pl
-----------------------------
#!/usr/bin/perl
# script: customizable.pl

use CGI::Pretty qw/:standard/;

my $color = param('color') || 'white';

print header,
    start_html({-bgcolor=>$color},'Customizable Page'),
    h1('Customizable Page'),
    "Set this page's background color to:",br,
    start_form,
    radio_group(-name=>'color',
                    -value=>['white','red','green','black',
                                'blue','silver','cyan'],
                    -cols=>2),
    submit(-name=>'Set Background'),
    end_form,
    p,
    hr,
    end_html;
```

1.2.2 Making Stateful Forms

Many real Web applications are more than a single page. Some may span multiple pages and fill-out forms. When the user goes from one page to the next, you've got to save the state of the previous page somewhere. A convenient and cheap place to put state information is in hidden fields in the form itself. Script I.2.1 is an example of a loan application with a total of five separate pages. Forward and back buttons allows the user to navigate between pages. The script remembers all the pages and summarizes them up at the end.

```
Script I.2.1: loan.pl
---------------------
#!/usr/local/bin/perl

# script: loan.pl
use CGI qw/:standard :html3/;

# this defines the contents of the fill out forms
# on each page.
my @PAGES = ('Personal Information','References','Assets','Review','Confirmation');
my %FIELDS = ('Personal Information' => ['Name','Address','Telephone','Fax'],
              'References'           => ['Personal Reference 1','Personal Reference 2'],
              'Assets'               => ['Savings Account','Home','Car']
             );
```

```perl
my %ALL_FIELDS = ();
# accumulate the field names into %ALL_FIELDS;
foreach (values %FIELDS) {
    grep($ALL_FIELDS{$_}++, @$_);
}


# figure out what page we're on and where we're heading.
my $current_page = calculate_page(param('page'),param('go'));
my $page_name = $PAGES[$current_page];

print_header($page_name);
print_form($current_page)         if $FIELDS{$page_name};
print_review($current_page)       if $page_name eq 'Review';
print_confirmation($current_page) if $page_name eq 'Confirmation';
print end_html;

# CALCULATE THE CURRENT PAGE
sub calculate_page {
    my ($prev, $dir) = @_;
    return 0 if $prev eq '';          # start with first page
    return $prev + 1 if $dir eq 'Submit Application';
    return $prev + 1 if $dir eq 'Next Page';
    return $prev - 1 if $dir eq 'Previous Page';
}

# PRINT HTTP AND HTML HEADERS
sub print_header {
    my $page_name = shift;
    print header,
    start_html("Your Friendly Family Loan Center"),
    h1("Your Friendly Family Loan Center"),
    h2($page_name);
}

# PRINT ONE OF THE QUESTIONNAIRE PAGES
sub print_form {
    my $current_page = shift;
    print "Please fill out the form completely and accurately.",
      start_form,
      hr;
    draw_form(@{$FIELDS{$page_name}});
    print hr;
    print submit(-name=>'go',-value=>'Previous Page')
        if $current_page > 0;
    print submit(-name=>'go',-value=>'Next Page'),
      hidden(-name=>'page',-value=>$current_page,-override=>1),
      end_form;
}

# PRINT THE REVIEW PAGE
sub print_review {
    my $current_page = shift;
    print "Please review this information carefully before submitting it. ",
      start_form;
    my (@rows);
    foreach $page ('Personal Information','References','Assets') {
        push(@rows,th({-align=>LEFT},em($page)));
```

```perl
        foreach $field (@{$FIELDS{$page}}) {
            push(@rows,
                TR(th({-align=>LEFT},$field),
                    td(param($field)))
                );
            print hidden(-name=>$field);
        }
    }

    print table({-border=>1},caption($page),@rows),
        hidden(-name=>'page',-value=>$current_page,-override=>1),
        submit(-name=>'go',-value=>'Previous Page'),
        submit(-name=>'go',-value=>'Submit Application'),
        end_form;
}

# PRINT THE CONFIRMATION PAGE
sub print_confirmation {
    print "Thank you. A loan officer will be contacting you shortly.",
        p,
        a({-href=>'../source.html'},'Code examples');
}


# CREATE A GENERIC QUESTIONNAIRE
sub draw_form {
    my (@fields) = @_;
    my (%fields);
    grep ($fields{$_}++, @fields);
    my (@hidden_fields) = grep(!$fields{$_}, keys %ALL_FIELDS);
    my (@rows);
    foreach (@fields) {
        push(@rows,
            TR(th({-align=>LEFT},$_),
                td(textfield(-name=>$_,-size=>50))
                )
            );
    }
    print table(@rows);

    foreach (@hidden_fields) {
        print hidden(-name=>$_);
    }
}
```

1.2.2.1 Keeping State with Cookies

If you want to maintain state even if the user quits the browser and comes back again, you can use cookies. Script I.2.2 records the user's name and color scheme preferences and recreates the page the way the user likes up to 30 days from the time the user last used the script.

```
Script I.2.2: preferences.pl
----------------------------
#!/usr/local/bin/perl

# file: preferences.pl
```

```perl
use CGI qw(:standard :html3);

# Some constants to use in our form.
my @colors = qw/aqua black blue fuschia gray green lime maroon navy olive
    purple red silver teal white yellow/;
my @sizes=("<default>",1..7);

# recover the "preferences" cookie.
my %preferences = cookie('preferences');

# If the user wants to change the background color or her
# name, they will appear among our CGI parameters.
foreach ('text','background','name','size') {
    $preferences{$_} = param($_) || $preferences{$_};
}

# Set some defaults
$preferences{'background'} ||= 'silver';
$preferences{'text'} ||= 'black';

# Refresh the cookie so that it doesn't expire.
my $the_cookie = cookie(-name=>'preferences',
                        -value=>\%preferences,
                        -path=>'/',
                        -expires=>'+30d');
print header(-cookie=>$the_cookie);

# Adjust the title to incorporate the user's name, if provided.
$title = $preferences{'name'} ?
    "Welcome back, $preferences{name}!" : "Customizable Page";

# Create the HTML page.  We use several of the HTML 3.2
# extended tags to control the background color and the
# font size.  It's safe to use these features because

# cookies don't work anywhere else anyway.
print start_html(-title=>$title,
                 -bgcolor=>$preferences{'background'},
                 -text=>$preferences{'text'}
                );

print basefont({-size=>$preferences{size}}) if $preferences{'size'} > 0;

print h1($title);

# Create the form
print hr,
    start_form,

    "Your first name: ",
    textfield(-name=>'name',
              -default=>$preferences{'name'},
              -size=>30),br,

    table(
          TR(
```

```
                 td("Preferred"),
                 td("Page color:"),
                 td(popup_menu(-name=>'background',
                               -values=>\@colors,
                               -default=>$preferences{'background'})
                    ),
                 ),
             TR(
                 td(''),
                 td("Text color:"),
                 td(popup_menu(-name=>'text',
                               -values=>\@colors,
                               -default=>$preferences{'text'})
                    )
                 ),
             TR(
                 td(''),
                 td("Font size:"),
                 td(popup_menu(-name=>'size',

                               -values=>\@sizes,
                               -default=>$preferences{'size'})
                    )
                 )
             ),
        submit(-label=>'Set preferences'),
        end_form,
        hr,
        end_html;
```

1.2.3 Creating Non-HTML Types

CGI can do more than just produce HTML documents. It can produce any type of document that you can output with Perl. This includes GIFs, Postscript files, sounds or whatever.

Script I.3.1 creates a clickable image map of a colored circle inside a square. The script is responsible both for generating the map and making the image (using the GD.pm library). It also creates a fill-out form that lets the user change the size and color of the image!

```
Script I.3.1: circle.pl
-----------------------
#!/usr/local/bin/perl

# script: circle.pl
use GD;
use CGI qw/:standard :imagemap/;

use constant RECTSIZE     => 100;
use constant CIRCLE_RADIUS  => 40;
my %COLORS = (
              'white' => [255,255,255],
              'red'   => [255,0,0],
              'green' => [0,255,0],
              'blue'  => [0,0,255],
```

```perl
                'black' => [0,0,0],
                'bisque'=> [255,228,196],
                'papaya whip' => [255,239,213],
                'sienna' => [160,82,45]
             );

my $draw          = param('draw');
my $circle_color  = param('color') || 'bisque';
my $mag           = param('magnification') || 1;

if ($draw) {
    draw_image();
} else {
    make_page();
}

sub draw_image {
    # create a new image
    my $im = new GD::Image(RECTSIZE*$mag,RECTSIZE*$mag);

    # allocate some colors
    my $white = $im->colorAllocate(@{$COLORS{'white'}});
    my $black = $im->colorAllocate(@{$COLORS{'black'}});
    my $circlecolor = $im->colorAllocate(@{$COLORS{$circle_color}});

    # make the background transparent and interlaced
    $im->transparent($white);
    $im->interlaced('true');

    # Put a black frame around the picture
    $im->rectangle(0,0,RECTSIZE*$mag-1,RECTSIZE*$mag-1,$black);

    # Draw the circle
    $im->arc(RECTSIZE*$mag/2,RECTSIZE*$mag/2,
             CIRCLE_RADIUS*$mag*2,
             CIRCLE_RADIUS*$mag*2,
             0,360,$black);

    # And fill it with circlecolor
    $im->fill(RECTSIZE*$mag/2,RECTSIZE*$mag/2,$circlecolor);

    # Convert the image to GIF and print it
    print header('image/gif'),$im->gif;
}

sub make_page {
   print header(),
   start_html(-title=>'Feeling Circular',-bgcolor=>'white'),
   h1('A Circle is as a Circle Does'),
   start_form,
   "Magnification: ",radio_group(-name=>'magnification',-values=>[1..4]),br,
   "Color: ",popup_menu(-name=>'color',-values=>[sort keys %COLORS]),
   submit(-value=>'Change'),
   end_form;
   print em(param('message') || 'click in the drawing' );

   my $url = url(-relative=>1,-query_string=>1);
```

```perl
    $url .= '?' unless param();
    $url .= '&draw=1';

    print p(
        img({-src=>$url,
             -align=>'LEFT',
             -usemap=>'#map',
             -border=>0}));


    print Map({-name=>'map'},
            Area({-shape=>'CIRCLE',
                -href=>param(-name=>'message',-value=>"You clicked in the circle")
                    && url(-relative=>1,-query_string=>1),
                -coords=>join(',',RECTSIZE*$mag/2,RECTSIZE*$mag/2,CIRCLE_RADIUS*$mag),
                -alt=>'Circle'}),
            Area({-shape=>'RECT',
                -href=>param(-name=>'message',-value=>"You clicked in the square")
                    && url(-relative=>1,-query_string=>1),
                -coords=>join(',',0,0,RECTSIZE*$mag,RECTSIZE*$mag),
                -alt=>'Square'}));
    print end_html;
}
```

Script I.3.2 creates a GIF89a animation. First it creates a set of simple GIFs, then uses the *combine* program (part of the ImageMagick package) to combine them together into an animation.

I'm not a good animator, so I can't do anything fancy. But you can!

```perl
Script I.3.2: animate.pl
-----------------------
#!/usr/local/bin/perl

# script: animated.pl
use GD;
use File::Path;

use constant START      => 80;
use constant END        => 200;
use constant STEP       => 10;
use constant COMBINE    => '/usr/local/bin/convert';
my @COMBINE_OPTIONS = (-delay => 5,
                       -loop  => 10000);

my @COLORS = ([240,240,240],
              [220,220,220],
              [200,200,200],
              [180,180,180],
              [160,160,160],
              [140,140,140],
              [150,120,120],
              [160,100,100],
              [170,80,80],
              [180,60,60],
              [190,40,40],
              [200,20,20],
              [210,0,0]);
```

```perl
@COLORS = (@COLORS,reverse(@COLORS));

my @FILES = ();
my $dir = create_temporary_directory();
my $index = 0;
for (my $r = START; $r <= END; $r+=STEP) {
    draw($r,$index,$dir);
    $index++;
}
for (my $r = END; $r > START; $r-=STEP) {
    draw($r,$index,$dir);
    $index++;

}

# emit the GIF89a
$| = 1;
print "Content-type: image/gif\n\n";
system COMBINE,@COMBINE_OPTIONS,@FILES,"gif:-";

rmtree([$dir],0,1);

sub draw {
    my ($r,$color_index,$dir) = @_;
    my $im = new GD::Image(END,END);
    my $white = $im->colorAllocate(255,255,255);
    my $black = $im->colorAllocate(0,0,0);
    my $color = $im->colorAllocate(@{$COLORS[$color_index % @COLORS]});
    $im->rectangle(0,0,END,END,$white);
    $im->arc(END/2,END/2,$r,$r,0,360,$black);
    $im->fill(END/2,END/2,$color);
    my $file = sprintf("%s/picture.%02d.gif",$dir,$color_index);
    open (OUT,">$file") || die "couldn't create $file: $!";
    print OUT $im->gif;
    close OUT;
    push(@FILES,$file);
}

sub create_temporary_directory {
    my $basename = "/usr/tmp/animate$$";
    my $counter=0;
    while ($counter < 100) {
        my $try = sprintf("$basename.%04d",$counter);
        next if -e $try;
        return $try if mkdir $try,0700;
    } continue { $counter++; }
    die "Couldn't make a temporary directory";
}
```

1.2.4 Document Translation

Did you know that you can use a CGI script to translate other documents on the fly? No s**t! Script I.4.1 is a script that intercepts all four-letter words in text documents and stars out the naughty bits. The document itself is specified using additional path information. We're a bit over-literal about what a four-letter word is, but what's the fun if you can't be extravagant?

```
Script I.4.1: naughty.pl

#!/usr/local/bin/perl
# Script: naughty.pl

use CGI ':standard';
my $file = path_translated() ||
            die "must be called with additional path info";
open (FILE,$file) || die "Can't open $file: $!\n";
print header('text/plain');
while (<FILE>) {
    s/\b(\w)\w{2}(\w)\b/$1**$2/g;
    print;
}
close FILE;
```

4.1 won't work on HTML files because the HTML tags will get starred out too. If you find it a little limiting to work only on plain-text files, script I.4.2 uses LWP's HTML parsing functions to modify just the text part of an HTML document without touching the tags. The script's a little awkward because we have to guess the type of file from the extension, and *redirect* when we're dealing with a non-HTML file. We can do better with *mod_perl*.

```
Script I.4.2: naughty2.pl
-------------------------
#!/usr/local/bin/perl

# Script: naughty2.pl
package HTML::Parser::FixNaughty;

require HTML::Parser;
@HTML::Parser::FixNaughty::ISA = 'HTML::Parser';

sub start {
    my ($self,$tag,$attr,$attrseq,$origtext) = @_;
    print $origtext;
}
sub end {
    my ($self,$tag) = @_;
    print "</$tag>";
}
sub text {
    my ($self,$text) = @_;
    $text =~ s/\b(\w)\w{2}(\w)\b/$1**$2/g;
    print $text;
}


package main;
use CGI qw/header path_info redirect path_translated/;

my $file = path_translated() ||
      die "must be called with additional path info";
$file .= "index.html" if $file =~ m!/$!;

unless ($file =~ /\.html?$/) {
    print redirect(path_info());
```

```
    exit 0;
}

my $parser = new HTML::Parser::FixNaughty;
print header();
$parser->parse_file($file);
```

A cleaner way to do this is to make this into an Apache Handler running under mod_perl. Let's look at `Apache::FixNaughty`:

```
file:Apache/FixNaughty.pm
-------------------------
# prefefine the HTML parser that we use afterwards
package HTML::Parser::FixNaughty;

require HTML::Parser;
@HTML::Parser::FixNaughty::ISA = 'HTML::Parser';

sub start {
    my ($self,$tag,$attr,$attrseq,$origtext) = @_;
    print $origtext;
}
sub end {
    my ($self,$tag) = @_;
    print "</$tag>";
}
sub text {
    my ($self,$text) = @_;
    $text =~ s/\b(\w)\w{2}(\w)\b/$1**$2/g;
    print $text;
}

# now for the mod_perl handler
package Apache::FixNaughty;

use Apache::Constants qw/:common/;
use strict;
use warnings;
use CGI qw/header path_info redirect path_translated/;

sub handler {
    my $r = shift;

    unless(-e $r->finfo) {
        $r->log_reason("Can't be found", $r->filename);
        return NOT_FOUND;
    }

    unless ($r->content_type eq 'text/html') {
        return DECLINED;

    }

    my $parser = new HTML::Parser::FixNaughty;

    $r->send_http_header('text/html');
    $parser->parse_file($file);
```

```
    return OK;
}

1;
__END__
```

You'll configure this like so:

```
Alias /naughty/ /path/to/doc/root/
<Location /naughty>
  SetHandler perl-script
  PerlHandler Apache::FixNaughty
</Location>
```

Now, all files being served below the */naughty* URL will be the same as those served from your document root, but will be processed and censured!

1.2.4.1 Smart Redirection

There's no need even to create a document with CGI. You can simply *redirect* to the URL you want. Script I.4.3 chooses a random picture from a directory somewhere and displays it. The directory to pick from is specified as additional path information, as in:

```
      /cgi-bin/random_pict/banners/egregious_advertising
```

```
Script I.4.3 random_pict.pl
--------------------------
#!/usr/local/bin/perl
# script: random_pict.pl

use CGI qw/:standard/;
my $PICTURE_PATH = path_translated();
my $PICTURE_URL = path_info();
chdir $PICTURE_PATH
        or die "Couldn't chdir to pictures directory: $!";
my @pictures = <*.{jpg,gif}>;
my $lucky_one = $pictures[rand(@pictures)];
die "Failed to pick a picture" unless $lucky_one;

print redirect("$PICTURE_URL/$lucky_one");
```

Under mod_perl, you would do this (the bigger size is because we're doing more checks here):

```
file:Apache/RandPicture.pm
--------------------------
package Apache::RandPicture;

use strict;
use Apache::Constants qw(:common REDIRECT);
use DirHandle ();

sub handler {
    my $r = shift;
    my $dir_uri = $r->dir_config('PictureDir');
```

```
    unless ($dir_uri) {
        $r->log_reason("No PictureDir configured");
        return SERVER_ERROR;
    }
    $dir_uri .= "/" unless $dir_uri =~ m:/$:;

    my $subr = $r->lookup_uri($dir_uri);
    my $dir = $subr->filename;
    # Get list of images in the directory.
    my $dh = DirHandle->new($dir);
    unless ($dh) {
        $r->log_error("Can't read directory $dir: $!");
        return SERVER_ERROR;
    }

    my @files;
    for my $entry ($dh->read) {
        # get the file's MIME type
        my $rr = $subr->lookup_uri($entry);
        my $type = $rr->content_type;
        next unless $type =~ m:^image/:;
        push @files, $rr->uri;
    }
    $dh->close;
    unless (@files) {
        $r->log_error("No image files in directory");
        return SERVER_ERROR;
    }

    my $lucky_one = $files[rand @files];
    # internal redirect, so we don't have to go back to the client
    $r->internal_redirect($lucky_one);
    return REDIRECT;
}

1;
__END__
```

1.2.5 File Uploads

Everyone wants to do it. I don't know why. Script I.5.1 shows a basic script that accepts a file to upload, reads it, and prints out its length and MIME type. Windows users should read about binmode() before they try this at home!

```
Script I.5.1 upload.pl
---------------------
#!/usr/local/bin/perl
#script: upload.pl

use CGI qw/:standard/;

print header,
    start_html('file upload'),
    h1('file upload');
print_form()    unless param;
```

```
print_results() if param;
print end_html;

sub print_form {
    print start_multipart_form(),
        filefield(-name=>'upload',-size=>60),br,
        submit(-label=>'Upload File'),
        end_form;
}

sub print_results {
    my $length;
    my $file = param('upload');
    if (!$file) {
        print "No file uploaded.";
        return;
    }
    print h2('File name'),$file;
    print h2('File MIME type'),
    uploadInfo($file)->{'Content-Type'};
    while (<$file>) {
        $length += length($_);
    }
    print h2('File length'),$length;
}
```

1.3 Part II: Web Site Care and Feeding

These scripts are designed to make your life as a Webmaster easier, leaving you time for more exciting things, like tango lessons.

1.3.1 Logs! Logs! Logs!

Left to their own devices, the log files will grow without limit, eventually filling up your server's partition and bringing things to a grinding halt. But wait! Don't turn off logging or throw them away. Log files are your friends.

1.3.1.1 Log rotation

Script II.1.1 shows the basic script for rotating log files. It renames the current "access_log" to "access_log.0", "access_log.0" to "access_log.1", and so on. The oldest log gets deleted. Run it from a cron job to keep your log files from taking over. The faster your log files grow, the more frequently you should run the script.

```
Script II.1.1: Basic Log File Rotation
--------------------------------------
#!/usr/local/bin/perl
$LOGPATH='/usr/local/apache/logs';
@LOGNAMES=('access_log','error_log','referer_log','agent_log');
$PIDFILE = 'httpd.pid';
$MAXCYCLE = 4;
```

```
chdir $LOGPATH;  # Change to the log directory
foreach $filename (@LOGNAMES) {
    for (my $s=$MAXCYCLE; $s >= 0; $s-- ) {
        $oldname = $s ? "$filename.$s" : $filename;
        $newname = join(".",$filename,$s+1);
        rename $oldname,$newname if -e $oldname;
    }
}
kill 'HUP',`cat $PIDFILE`;
```

1.3.1.2 Log rotation and archiving

But some people don't want to delete the old logs. Wow, maybe some day you could sell them for a lot of money to a marketing and merchandising company! Script II.1.2 appends the oldest to a gzip archive. Log files compress extremely well and make great bedtime reading.

```
Script II.1.2: Log File Rotation and Archiving
----------------------------------------------
#!/usr/local/bin/perl
$LOGPATH    = '/usr/local/apache/logs';
$PIDFILE    = 'httpd.pid';
$MAXCYCLE   = 4;
$GZIP       = '/bin/gzip';

@LOGNAMES=('access_log','error_log','referer_log','agent_log');
%ARCHIVE=('access_log'=>1,'error_log'=>1);

chdir $LOGPATH;  # Change to the log directory
foreach $filename (@LOGNAMES) {
  system "$GZIP -c $filename.$MAXCYCLE >> $filename.gz"
        if -e "$filename.$MAXCYCLE" and $ARCHIVE{$filename};
   for (my $s=$MAXCYCLE; $s >= 0; $s-- ) {
        $oldname = $s ? "$filename.$s" : $filename;
        $newname = join(".",$filename,$s+1);
        rename $oldname,$newname if -e $oldname;
    }
}
kill 'HUP',`cat $PIDFILE`;
```

1.3.1.3 Log rotation, compression and archiving

What's that? Someone broke into your computer, stole your log files and now **he's** selling it to a Web marketing and merchandising company? Shame on them. And on you for letting it happen. Script II.1.3 uses *idea* (part of the SSLEay package) to encrypt the log before compressing it. You need GNU tar to run this one. The log files are individually compressed and encrypted, and stamped with the current date.

```
Script II.1.3: Log File Rotation and Encryption
-----------------------------------------------
#!/usr/local/bin/perl
use POSIX 'strftime';

$LOGPATH     = '/home/www/logs';
$PIDFILE     = 'httpd.pid';
$MAXCYCLE    = 4;
```

```
$IDEA        = '/usr/local/ssl/bin/idea';
$GZIP        = '/bin/gzip';
$TAR         = '/bin/tar';
$PASSWDFILE  = '/home/www/logs/secret.passwd';

@LOGNAMES=('access_log','error_log','referer_log','agent_log');
%ARCHIVE=('access_log'=>1,'error_log'=>1);

chdir $LOGPATH;  # Change to the log directory
foreach $filename (@LOGNAMES) {
    my $oldest = "$filename.$MAXCYCLE";
    archive($oldest) if -e $oldest and $ARCHIVE{$filename};
    for (my $s=$MAXCYCLE; $s >= 0; $s-- ) {
        $oldname = $s ? "$filename.$s" : $filename;
        $newname = join(".",$filename,$s+1);
        rename $oldname,$newname if -e $oldname;
    }
}
kill 'HUP','cat $PIDFILE';

sub archive {
    my $f = shift;
    my $base = $f;
    $base =~ s/\.\d+$//;
    my $fn = strftime("$base.%Y-%m-%d_%H:%M.gz.idea",localtime);
    system "$GZIP -9 -c $f | $IDEA -kfile $PASSWDFILE > $fn";
    system "$TAR rvf $base.tar --remove-files $fn";
}
```

1.3.1.4 Log Parsing

There's a lot you can learn from log files. Script II.1.4 does the basic access log regular expression match. What you do with the split-out fields is limited by your imagination. Here's a typical log entry so that you can follow along (wrapped for readability):

```
portio.cshl.org - - [03/Feb/1998:17:42:15 -0500]
    "GET /pictures/small_logo.gif HTTP/1.0" 200 2172

Script II.1.4: Basic Log Parsing
-------------------------------
#!/usr/local/bin/perl

$REGEX=/^(\S+) (\S+) (\S+) \[([^]]+)\] "(\w+) (\S+).*" (\d+) (\S+)/;
while (<>) {
    ($host,$rfc931,$user,$date,$request,$URL,$status,$bytes) = m/$REGEX/o;
     &collect_some_statistics;
}
&print_some_statistics;

sub collect_some_statistics {
  # for you to fill in
}

sub print_some_statistics {
  # for you to fill in
}
```

Script II.1.5 scans the log for certain status codes and prints out the top URLs or hosts that triggered them. It can be used to get quick-and-dirty usage statistics, to find broken links, or to detect certain types of breakin attempts. Use it like this:

```
% find_status.pl -t10 200 ~www/logs/access_log

 TOP 10 URLS/HOSTS WITH STATUS CODE 200:

   REQUESTS  URL/HOST
   --------  --------
     1845    /www/wilogo.gif
     1597    /cgi-bin/contig/sts_by_name?database=release
     1582    /WWW/faqs/www-security-faq.html
     1263    /icons/caution.xbm
      930    /
      886    /ftp/pub/software/WWW/cgi_docs.html
      773    /cgi-bin/contig/phys_map
      713    /icons/dna.gif
      686    /WWW/pics/small_awlogo.gif

 Script II.1.5: Find frequent status codes
 -----------------------------------------
 #!/usr/local/bin/perl
 # File: find_status.pl

 require "getopts.pl";
 &Getopts('L:t:h') || die <<USAGE;
 Usage: find_status.pl [-Lth] <code1> <code2> <code3> ...
        Scan Web server log files and list a summary
        of URLs whose requests had the one of the
        indicated status codes.
 Options:
        -L <domain>  Ignore local hosts matching this domain
        -t <integer> Print top integer URLS/HOSTS [10]
        -h           Sort by host rather than URL
 USAGE
     ;
 if ($opt_L) {
     $opt_L=~s/\./\\./g;
     $IGNORE = "(^[^.]+|$opt_L)\$";
 }
 $TOP=$opt_t || 10;

 while (@ARGV) {
     last unless $ARGV[0]=~/^\d+$/;
     $CODES{shift @ARGV}++;
 }

 while (<>) {
     ($host,$rfc931,$user,$date,$request,$URL,$status,$bytes) =
         /^(\S+) (\S+) (\S+) \[([^]]+)\] "(\w+) (\S+).*" (\d+) (\S+)/;
     next unless $CODES{$status};
     next if $IGNORE && $host=~/$IGNORE/io;
     $info = $opt_h ? $host : $URL;
     $found{$status}->{$info}++;
 }
```

```
foreach $status (sort {$a<=>$b;} sort keys %CODES) {
    $info = $found{$status};
    $count = $TOP;

    foreach $i (sort {$info->{$b} <=> $info->{$a};} keys %{$info}) {
        write;
        last unless --$count;
    }
    $- = 0;  # force a new top-of-report
}

format STDOUT_TOP=

TOP @## URLS/HOSTS WITH STATUS CODE @##:
    $TOP,                       $status

    REQUESTS  URL/HOST
    --------  --------
.
format STDOUT=
    @#####    @<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
    $info->{$i},$i
.
```

1.3.1.5 Offline Reverse DNS Resolution

Many sites turn off reverse name look-ups in order to improve server performance. The log files will
contain the IP addresses of remote hosts, but not their DNS names. Script II.1.6 will do the reverse name
resolution off-line. You can run it before the log rotation and archiving scripts, preferably on a machine
that isn't busy serving Web requests at the same time.

This script maintains a cache of resolved names. Because performance is more important than complete-
ness, if an address doesn't resolve after two seconds, it moves on to the next one and never tries that name
again.

```
Script II.1.6: Reverse DNS Resolution
-------------------------------------
#!/usr/local/bin/perl

use constant TIMEOUT => 2;
$SIG{ALRM} = sub {die "timeout"};

while (<>) {
    s/^(\S+)/lookup($1)/e;
} continue {
    print;
}

sub lookup {
    my $ip = shift;
    return $ip unless $ip=~/\d+\.\d+\.\d+\.\d+/;
    return $CACHE{$ip} if exists $CACHE{$ip};
    my @h = eval <<'END';
    alarm(TIMEOUT);
```

```
    my @i = gethostbyaddr(pack('C4',split('\.',$ip)),2);
    alarm(0);
    @i;
END
    $CACHE{$ip} = $h[0];
    return $CACHE{$ip} || $ip;
}
```

1.3.1.6 Detecting Robots

I was very upset a few months ago when I did some log analysis and discovered that 90% of my hits were coming from 10% of users, and that those 10% were all robots! Script II.1.7 is the script I used to crunch the log and perform the analysis. The script works like this:

1. we assume that anyone coming from the same IP address with the same user agent within 30 minutes is the same person/robot (not quite right, but close enough).

2. anything that fetches /robots.txt is probably a robot, and a "polite" one, to boot.

3. we count the total number of accesses a user agent makes.

4. we average the interval between successive fetches.

5. we calculate an "index" which is the number of hits over the interval. Robots have higher indexes than people.

6. we print everything out in a big tab-delimited table for graphing.

By comparing the distribution of "polite" robots to the total distribution, we can make a good guess as to who the impolite robots are.

```
Script II.1.7: Robo-Cop
-----------------------
#!/usr/local/bin/perl

use Time::ParseDate;
use strict 'vars';

# after 30 minutes, we consider this a new session
use constant MAX_INTERVAL => 60*30;
my (%HITS,%INT_NUMERATOR,%INT_DENOMINATOR,%POLITE,%LAST,$HITS);

# This uses a non-standard agent log with lines formatted like this:
# [08/Feb/1998:12:28:35 -0500] phila249-pri.voicenet.com "Mozilla/3.01 (Win95; U)" /cgi-bin/fortune

my $file = shift;
open (IN,$file=~/\.gz$/ ? "zcat $file |" : $file ) || die "Can't open file/pipe: $!";

while (<IN>) {
    my($date,$host,$agent,$URL) = /^\[(.+)\] (\S+) "(.*)" (\S+)$/;
    next unless $URL=~/\.(html|htm|txt)$/;

    $HITS++;
    $host = "$host:$agent"; # concatenate host and agent
    $HITS{$host}++;
    my $seconds = parsedate($date);
    if ($LAST{$host}) {
```

```
        my $interval = $seconds - $LAST{$host};
         if ($interval < MAX_INTERVAL) {
             $INT_NUMERATOR{$host} += $interval;
             $INT_DENOMINATOR{$host}++;
         }
     }
     $LAST{$host} = $seconds;
     $POLITE{$host}++ if $URL eq '/robots.txt';
     print STDERR $HITS,"\n" if ($HITS % 1000) == 0;
}

# print out, sorted by hits
print join("\t",qw/Client Robot Hits Interval Hit_Percent Index/),"\n";
foreach (sort {$HITS{$b}<=>$HITS{$a}} keys %HITS) {

    next unless $HITS{$_} >= 5;              # not enough total hits to mean much
    next unless $INT_DENOMINATOR{$_} >= 5;  # not enough consecutive hits to mean much

    my $mean_interval = $INT_NUMERATOR{$_}/$INT_DENOMINATOR{$_};
    my $percent_hits = 100*($HITS{$_}/$HITS);
    my $index = $percent_hits/$mean_interval;

    print join("\t",
               $_,
               $POLITE{$_} ? 'yes' : 'no',
               $HITS{$_},
               $mean_interval,
               $percent_hits,
               $index
              ),"\n";
}
```

1.3.1.7 Logging to syslog

If you run a large site with many independent servers, you might be annoyed that they all log into their own file systems rather than into a central location. Apache offers a little-known feature that allows it to send its log entries to a process rather than a file. The process (a Perl script, natch) can do whatever it likes with the logs. For instance, using Tom Christiansen's `Syslog` module to send the info to a remote syslog daemon.

Here's what you add to the Apache httpd.conf file:

```
<VirtualHost www.company1.com>
   CustomLog "| /usr/local/apache/bin/logger company1" common
   # blah blah
</VirtualHost>

<VirtualHost www.company2.com>
   CustomLog "| /usr/local/apache/bin/logger company2" common
   # blah blah
</VirtualHost>
```

Do the same for each server on the local network.

Here's what you add to each Web server's syslog.conf (this assumes that the central logging host has the alias hostname "loghost":

```
  local0.info                           @loghost
```

Here's what you add to the central log host's syslog.conf:

```
  local0.info                           /var/log/web/access_log
```

Script II.1.8 shows the code for the "logger" program:

```
  Script II.1.8 "logger"
  ---------------------
  #!/usr/local/bin/perl
  # script: logger

  use Sys::Syslog;

  $SERVER_NAME = shift || 'www';
  $FACILITY = 'local0';
  $PRIORITY = 'info';

  Sys::Syslog::setlogsock('unix');
  openlog ($SERVER_NAME,'ndelay',$FACILITY);
  while (<>) {
      chomp;
      syslog($PRIORITY,$_);
  }
  closelog;
```

1.3.1.8 Logging to a relational database

One of the selling points of the big commercial Web servers is that they can log to relational databases via ODBC. Big whoop. With a little help from Perl, Apache can do that too. Once you've got the log in a relational database, you can data mine to your heart's content.

This example uses the freeware mySQL DBMS. To prepare, create an appropriate database containing a table named "access_log". It should have a structure like this one. Add whatever indexes you think you need. Also notice that we truncate URLs at 255 characters. You might want to use TEXT columns instead.

```
  CREATE TABLE access_log (
        when    datetime    not null,
        host        varchar(255) not null,
        method  char(4)      not null,
        url         varchar(255) not null,
        auth        varchar(50),
        browser varchar(50),
        referer varchar(255),
        status  smallint(3)  not null,
        bytes       int(8)       default 0
  );
```

Now create the following entries in httpd.conf:

```
LogFormat "\"%{%Y-%m-%d %H:%M:%S}t\" %h \"%r\" %u \"%{User-agent}i\" %{Referer}i %s %b" mysql
CustomLog "| /usr/local/apache/bin/mysqllog" mysql
```

Script II.1.9 is the source code for mysqllog.

```
Script II.1.9 "mysqllog"
-----------------------
#!/usr/local/bin/perl
# script: mysqllog
use DBI;

use constant DSN       => 'dbi:mysql:www';
use constant DB_TABLE  => 'access_log';
use constant DB_USER   => 'nobody';
use constant DB_PASSWD => '';

$PATTERN = '"([^"]+)" (\S+) "(\S+) (\S+) [^"]+" (\S+) "([^"]+)" (\S+) (\d+) (\S+)';

$db = DBI->connect(DSN,DB_USER,DB_PASSWD) || die DBI->errstr;
$sth = $db->prepare("INSERT INTO ${\DB_TABLE} VALUES(?,?,?,?,?,?,?,?,?)")
    || die $db->errstr;
while (<>) {
    chomp;
    my($date,$host,$method,$url,$user,$browser,$referer,$status,$bytes) = /$PATTERN/o;
    $user    = undef if $user    eq '-';
    $referer = undef if $referer eq '-';
    $browser = undef if $browser eq '-';
    $bytes   = undef if $bytes   eq '-';
    $sth->execute($date,$host,$method,$url,$user,$browser,$referer,$status,$bytes);
}
$sth->finish;
$db->disconnect;
```

NOTE: Your database will grow very quickly. Make sure that you have a plan for truncating or archiving the oldest entries. Or have a lot of storage space handy! Also be aware that this will cause a lot of traffic on your LAN. Better start shopping around for 100BT hubs.

1.3.2 My server fell down and it can't get up!

Web servers are very stable and will stay up for long periods of time if you don't mess with them. However, human error can bring them down, particularly if you have a lot of developers and authors involved in running the site. The scripts in this section watch the server and send you an email message when there's a problem.

1.3.2.1 Monitoring a local server

The simplest script just tries to signal the Web server process. If the process has gone away, it sends out an S.O.S. See script II.2.1 shows the technique. Notice that the script has to run as *root* in order to successfully signal the server.

```
Script II.2.1 "localSOS"
------------------------
#!/usr/local/bin/perl
# script: localSOS

use constant PIDFILE  => '/usr/local/apache/var/run/httpd.pid';
$MAIL               = '/usr/sbin/sendmail';
$MAIL_FLAGS         = '-t -oi';
$WEBMASTER          = 'webmaster';

open (PID,PIDFILE) || die PIDFILE,": $!\n";
$pid = <PID>;  close PID;
kill 0,$pid || sos();

sub sos {
  open (MAIL,"| $MAIL $MAIL_FLAGS") || die "mail: $!";
  my $date = localtime();
  print MAIL <<END;
To: $WEBMASTER
From: The Watchful Web Server Monitor <nobody>
Subject: Web server is down

I tried to call the Web server at $date but there was
no answer.

Respectfully yours,

The Watchful Web Server Monitor
END
  close MAIL;
}
```

1.3.2.2 Monitoring a remote server

Local monitoring won't catch problems with remote machines, and they'll miss subtle problems that can happen when the Web server hangs but doesn't actually crash. A functional test is better. Script II.2.2 uses the LWP library to send a HEAD request to a bunch of servers. If any of them fails to respond, it sends out an SOS. This script does **not** have to run as a privileged user.

```
Script II.2.2 "remoteSOS"
-------------------------
#!/usr/local/bin/perl
# script: remoteSOS

use LWP::Simple;
%SERVERS = (
       "Fred's server"   => 'http://www.fred.com',
       "Martha's server" => 'http://www.stewart-living.com',
       "Bill's server"   => 'http://www.whitehouse.gov'
       );
$MAIL               = '/usr/sbin/sendmail';
$MAIL_FLAGS         = '-t -oi';
$WEBMASTER          = 'webmaster';

foreach (sort keys %SERVERS) {
   sos($_) unless head($SERVERS{$_});
```

```
  }

  sub sos {
    my $server = shift;
    open (MAIL,"| $MAIL $MAIL_FLAGS") || die "mail: $!";
    my $date = localtime();
    print MAIL <<END;
To: $WEBMASTER
From: The Watchful Web Server Monitor <nobody>
Subject: $server is down

I tried to call $server at $date but there was
no one at home.

Respectfully yours,

The Watchful Web Server Monitor
END
    close MAIL;
  }
```

1.3.2.3 Resurrecting Dead Servers

So it's not enough to get e-mail that the server's down, you want to relaunch it as well? Script II.2.3 is a
hybrid of localSOS and remoteSOS that tries to relaunch the local server after sending out the SOS. It has
to be run as **root**, unless you've made *apachectl* suid to root.

```
  Script II.2.2 "webLazarus"
  ------------------------
  #!/usr/local/bin/perl
  # script: webLazarus

  use LWP::Simple;
  use constant URL        => 'http://presto.capricorn.com/';
  use constant APACHECTL => '/usr/local/apache/bin/apachectl';
  $MAIL                   = '/usr/sbin/sendmail';
  $MAIL_FLAGS             = '-t -oi';
  $WEBMASTER              = 'lstein@prego.capricorn.com';

  head(URL) || resurrect();

  sub resurrect {
      open (STDOUT,"| $MAIL $MAIL_FLAGS") || die "mail: $!";
      select STDOUT; $| = 1;
      open (STDERR,">&STDOUT");

      my $date = localtime();
      print <<END;
To: $WEBMASTER
From: The Watchful Web Server Monitor <nobody>
Subject: Web server is down

I tried to call the Web server at $date but there was
no answer.  I am going to try to resurrect it now:

Mumble, mumble, mumble, shazzzzammmm!
```

```
END
    ;

    system APACHECTL,'restart';

    print <<END;

That's the best I could do.  Hope it helped.

Worshipfully yours,


The Web Monitor
END
    close STDERR;
    close STDOUT;
}
```

Here's the message you get when the script is successful:

```
Date: Sat, 4 Jul 1998 14:55:38 -0400
To: lstein@prego.capricorn.com
Subject: Web server is down

I tried to call the Web server at Sat Jul  4 14:55:37 1998 but there was
no answer.  I am going to try to resurrect it now:

Mumble, mumble, mumble, shazzzzammmm!

/usr/local/apache/bin/apachectl restart: httpd not running, trying to start
[Sat Jul  4 14:55:38 1998] [debug] mod_so.c(258): loaded module setenvif_module
[Sat Jul  4 14:55:38 1998] [debug] mod_so.c(258): loaded module unique_id_module
/usr/local/apache/bin/apachectl restart: httpd started

That's the best I could do.  Hope it helped.

Worshipfully yours,

The Web Monitor
```

1.3.3 Site Replication and Mirroring

Often you will want to mirror a page or set of pages from another server, for example, to distribute the load amongst several replicate servers, or to keep a set of reference pages handy. The LWP library makes this easy.

1.3.3.1 Mirroring Single Pages

```
% ./MirrorOne.pl
cats.html: Not Modified
dogs.html: OK
gillie_fish.html: Not Modified
```

```
Script II.3.1 mirrorOne.pl
--------------------------
#!/usr/local/bin/perl
# mirrorOne.pl

use LWP::Simple;
use HTTP::Status;

use constant DIRECTORY => '/local/web/price_lists';
%DOCUMENTS = (
        'dogs.html'  => 'http://www.pets.com/dogs/price_list.html',
        'cats.html'  => 'http://www.pets.com/cats/price_list.html',
        'gillie_fish.html' => 'http://aquaria.com/prices.html'
        );
chdir DIRECTORY;
foreach (sort keys %DOCUMENTS) {
   my $status = mirror($DOCUMENTS{$_},$_);
   warn "$_: ",status_message($status),"\n";
}
```

1.3.3.2 Mirroring a Document Tree

With a little more work, you can recursively mirror an entire set of linked pages. Script II.3.2 mirrors the requested document and all subdocuments, using the LWP HTML::LinkExtor module to extract all the HTML links.

```
Script II.3.2 mirrorTree.pl
--------------------------
#!/usr/local/bin/perl

# File: mirrorTree.pl

use LWP::UserAgent;
use HTML::LinkExtor;
use URI::URL;
use File::Path;
use File::Basename;
%DONE    = ();

my $URL = shift;

$UA     = new LWP::UserAgent;
$PARSER = HTML::LinkExtor->new();
$TOP    = $UA->request(HTTP::Request->new(HEAD => $URL));
$BASE   = $TOP->base;

mirror(URI::URL->new($TOP->request->url));

sub mirror {
   my $url = shift;

   # get rid of query string "?" and fragments "#"
   my $path = $url->path;
   my $fixed_url = URI::URL->new ($url->scheme . '://' . $url->netloc . $path);

   # make the URL relative
```

```perl
    my $rel = $fixed_url->rel($BASE);
    $rel .= 'index.html' if $rel=~m!/$! || length($rel) == 0;

    # skip it if we've already done it
    return if $DONE{$rel}++;

    # create the directory if it doesn't exist already
    my $dir = dirname($rel);
    mkpath([$dir]) unless -d $dir;

    # mirror the document
    my $doc = $UA->mirror($fixed_url,$rel);
    print STDERR "$rel: ",$doc->message,"\n";
    return if $doc->is_error;

    # Follow HTML documents
    return unless $rel=~/\.html?$/i;
    my $base = $doc->base;

    # pull out the links and call us recursively
    my @links = $PARSER->parse_file("$rel")->links;
    my @hrefs = map { url($_->[2],$base)->abs } @links;

    foreach (@hrefs) {
        next unless is_child($BASE,$_);
         mirror($_);
    }

}

sub is_child {
    my ($base,$url) = @_;
    my $rel = $url->rel($base);
    return ($rel ne $url) && ($rel !~ m!^[/.]!);
}
```

1.3.3.3 Checking for Bad Links

A slight modification of this last script allows you to check an entire document hierarchy (your own or someone else's) for bad links. The script shown in II.3.3 traverses a document, and checks each of the http:, ftp: and gopher: links to see if there's a response at the other end. Links that point to sub-documents are fetched and traversed as before, so you can check your whole site in this way.

```
% find_bad_links http://prego/apache-1.2/
checking http://prego/apache-1.2/...
checking http://prego/apache-1.2/manual/...
checking http://prego/apache-1.2/manual/misc/footer.html...
checking http://prego/apache-1.2/manual/misc/header.html...
checking http://prego/apache-1.2/manual/misc/nopgp.html...
checking http://www.yahoo.com/Science/Mathematics/Security_and_Encryption/...
checking http://www.eff.org/pub/EFF/Policy/Crypto/...
checking http://www.quadralay.com/www/Crypt/Crypt.html...
checking http://www.law.indiana.edu/law/iclu.html...
checking http://bong.com/~brian...
checking http://prego/apache-1.2/manual/cgi_path.html...
checking http://www.ics.uci.edu/pub/ietf/http/...
```

```
       .
       .
       .
 BAD LINKS:
 manual/misc/known_bugs.html : http://www.apache.org/dist/patches/apply_to_1.2b6/
 manual/misc/fin_wait_2.html : http://www.freebsd.org/
 manual/misc/fin_wait_2.html : http://www.ncr.com/
 manual/misc/compat_notes.html : http://www.eit.com/
 manual/misc/howto.html : http://www.zyzzyva.com/robots/alert/
 manual/misc/perf.html : http://www.software.hp.com/internet/perf/tuning.html
 manual/misc/perf.html : http://www.qosina.com/~awm/apache/linux-tcp.html
 manual/misc/perf.html : http://www.sun.com/sun-on-net/Sun.Internet.Solutions/performance/
 manual/misc/perf.html : http://www.sun.com/solaris/products/siss/
 manual/misc/nopgp.html : http://www.yahoo.com/Science/Mathematics/Security_and_Encryption/

 152 documents checked
 11 bad links

 Script II.3.2 find_bad_links.pl
 -------------------------------
 #!/usr/local/bin/perl

 # File: find_bad_links.pl

 use LWP::UserAgent;
 use HTML::LinkExtor;
 use URI::URL;
 use WWW::RobotRules;

 %CAN_HANDLE = ('http'=>1,
                'gopher'=>1,
                # 'ftp'=>1,   # timeout problems?
                );
 %OUTCOME = ();
 $CHECKED = $BAD = 0;
 @BAD = ();

 my $URL = shift;

 $UA     = new LWP::UserAgent;
 $PARSER = HTML::LinkExtor->new();
 $TOP    = $UA->request(HTTP::Request->new(HEAD => $URL));
 $BASE   = $TOP->base;

 # handle robot rules
 my $robots = URI::URL->new('robots.txt',$BASE->scheme.'://'.$BASE->netloc);
 my $robots_text = $UA->request(HTTP::Request->new(GET=>$robots))->content;
 $ROBOTRULES = WWW::RobotRules->new;
 $ROBOTRULES->parse($robots->abs,$robots_text);

 check_links(URI::URL->new($TOP->request->url));
 if (@BAD) {
     print "\nBAD LINKS:\n";
     print join("\n",@BAD),"\n\n";
 }
 print "$CHECKED documents checked\n",scalar(@BAD)," bad links\n";

 sub check_links {
```

```
    my $url = shift;
    my $fixed_url = $url;
    $fixed_url =~ s/\#.+$//;

    return 1 unless $CAN_HANDLE{$url->scheme};

    # check cached outcomes
    return $OUTCOME{$fixed_url} if exists $OUTCOME{$fixed_url};

    print STDERR "checking $fixed_url...\n";
    $CHECKED++;

    my $rel = $url->rel($BASE) || 'index.html';
    my $child = is_child($BASE,$url);
    $UA->timeout(5);
    my $doc = $d = $UA->request(HTTP::Request->new(($child ? 'GET' : 'HEAD' )=>$url));
    $OUTCOME{$fixed_url} = $doc->is_success;

    return $OUTCOME{$fixed_url}
    unless $ROBOTRULES->allowed($fixed_url)
      && $child && $doc->header('Content-type') eq 'text/html';

    # Follow HTML documents
    my $base = $doc->base;

    # pull out the links and call us recursively
    my @links = $PARSER->parse($doc->content)->links;
    my @hrefs = map { url($_->[2],$base)->abs } @links;

    foreach (@hrefs) {
        next if check_links($_);
        push (@BAD,"$rel : $_");
    }
    1;
}

sub is_child {
    my ($base,$url) = @_;
    my $rel = $url->rel($base);
    return ($rel ne $url) && ($rel !~ m!^[/.]!);

}
```

1.3.4 Load balancing

You've hit the big time, and your site is getting more hits than you ever dreamed of. Millions, zillions of hits. What's that? System load just passed 50 and response time is getting kinda' s-l-o-w-w-w?

Perl to the rescue. Set up several replica Web servers with different hostnames and IP addresses. Run this script on the "main" site and watch it round-robin the requests to the replica servers. It uses IO::Socket to listen for incoming requests on port 80. It then changes its privileges to run as nobody.nogroup, just like a real Web server. Next it preforks itself a few times (and you always thought preforking was something fancy, didn't you?), and goes into an accept() loop. Each time an incoming session comes in, it forks off another child to handle the request. The child reads the HTTP request and issues the an HTTP redirec-

tion to send the browser to a randomly selected server.

NOTE: Another way to do this is to have multiple "A" records defined for your server's hostname and let DNS caching distribute the load.

```
Script II.4.1: A Load Balancing "Web Server"
--------------------------------------------
#!/usr/local/bin/perl

# list of hosts to balance between
@HOSTS = qw/www1.web.org www2.web.org www3.web.org www4.web.org/;

use IO::Socket;
$SIG{CHLD} = sub { wait() };
$ENV{'PATH'}='/bin:/usr/bin';
chomp($hostname = '/bin/hostname');

# Listen on port 80
$sock = IO::Socket::INET->new(Listen  => 5,
                             LocalPort => 80,
                             LocalAddr => $hostname,
                             Reuse     => 1,
                             Proto     => 'tcp');

# become "nobody"
$nobody  = (getpwnam('nobody'))[2]  || die "nobody is nobody";
$nogroup = (getgrnam('nogroup'))[2] || die "can't grok nogroup";
($<,$() = ($>,$)) = ($nobody,$nogroup); # get rid of root privileges!
($\,$/) = ("\r\n","\r\n\r\n");          # CR/LF on output/input

# Go into server mode
close STDIN; close STDOUT; close STDERR;

# prefork -- gee is that all there is to it?
fork() && fork() && fork() && fork() && exit 0;

# start accepting connections
while (my $s = $sock->accept()) {
    do { $s->close; next; } if fork();
    my $request = <$s>;
    redirect($1,$s) if $request=~/(?:GET|POST|HEAD|PUT)\s+(\S+)/;
    $s->flush;
    undef $s;
    exit 0;
}


sub redirect {
    my ($url,$s) = @_;
    my $host = $HOSTS[rand(@HOSTS)];
    print $s "HTTP/1.0 301 Moved Temporarily";
    print $s "Server: Lincoln's Redirector/1.0";
    print $s "Location: http://${host}${url}";
    print $s "";
}
```

1.3.5 Torture Testing a Server

Any server written in C suffers the risk of static buffer overflow bugs. In the past, these bugs have led to security compromises and Web server breakins. Script II.2.3 torture tests servers and CGI scripts by sending large amounts of random date to them. If the server crashes, it probably contains a buffer overflow bug.

Here's what you see when a server crashes:

```
% torture.pl -t 1000 -l 5000 http://www.capricorn.com
torture.pl version 1.0 starting
Base URL:               http://www.capricorn.com/cgi-bin/search
Max random data length: 5000
Repetitions:            1000
Post:                   0
Append to path:         0
Escape URLs:            0

200 OK
200 OK
200 OK
200 OK
200 OK
500 Internal Server Error
500 Could not connect to www.capricorn.com:80
500 Could not connect to www.capricorn.com:80
500 Could not connect to www.capricorn.com:80

Script II.5.1: torture tester
-----------------------------
#!/usr/local/bin/perl

# file: torture.pl
# Torture test Web servers and scripts by sending them large arbitrary URLs
# and record the outcome.

use LWP::UserAgent;
use URI::Escape 'uri_escape';
require "getopts.pl";

$USAGE = <<USAGE;
Usage: $0 -[options] URL
Torture-test Web servers and CGI scripts

Options:
 -l <integer>  Max length of random URL to send [1024 bytes]
 -t <integer>  Number of times to run the test [1]
 -P            Use POST method rather than GET method
 -p            Attach random data to path rather than query string
 -e            Escape the query string before sending it
USAGE

$VERSION = '1.0';

# process command line
```

```
  &Getopts('l:t:Ppe') || die $USAGE;

  # get parameters
  $URL    = shift || die $USAGE;
  $MAXLEN = $opt_l ne '' ? $opt_l : 1024;
  $TIMES  = $opt_t || 1;
  $POST   = $opt_P || 0;
  $PATH   = $opt_p || 0;
  $ESCAPE = $opt_e || 0;

  # cannot do both a post and a path at the same time
  $POST = 0 if $PATH;

  # create an LWP agent
  my $agent = new LWP::UserAgent;

  print <<EOF;
  torture.pl version $VERSION starting
  Base URL:               $URL
  Max random data length: $MAXLEN
  Repetitions:            $TIMES
  Post:                   $POST
  Append to path:         $PATH
  Escape URLs:            $ESCAPE

  EOF


  # Do the test $TIMES times
  while ($TIMES) {
      # create a string of random stuff
      my $garbage = random_string(rand($MAXLEN));
      $garbage = uri_escape($garbage) if $ESCAPE;
      my $url = $URL;
      my $request;

      if (length($garbage) == 0) { # if no garbage to add, just fetch URL
          $request = new HTTP::Request ('GET',$url);
      }

      elsif ($POST) {                 # handle POST request
         my $header = new HTTP::Headers (
                                         Content_Type => 'application/x-www-form-urlencoded',
                                         Content_Length => length($garbage)
                                         );
        # garbage becomes the POST content
        $request = new HTTP::Request ('POST',$url,$header,$garbage);

      } else {                        # handle GET request

        if ($PATH) {                # append garbage to the base URL
            chop($url) if substr($url,-1,1) eq '/';
            $url .= "/$garbage";

        } else {                    # append garbage to the query string
            $url .= "?$garbage";
        }

        $request = new HTTP::Request ('GET',$url);
```

```
    }

    # do the request and fetch the response
    my $response = $agent->request($request);

    # print the numeric response code and the message
    print $response->code,' ',$response->message,"\n";

} continue { $TIMES-- }

# return some random data of the requested length
sub random_string {
    my $length = shift;
    return undef unless $length >= 1;
    return join('',map chr(rand(255)),0..$length-1);
}
```

For other load testing tools, have a look at our Benchmarking section.

1.4 Part III: mod_perl -- Faster Than a Speeding Bullet

mod_perl is Doug MacEachern's embedded Perl for Apache. With a mod_perl-enabled server, there's no tedious waiting around while the Perl interpreter fires up, reads and compiles your script. It's right there, ready and waiting. What's more, once compiled your script remains in memory, all charged and raring to go. Suddenly those sluggish Perl CGI scripts race along at compiled C speeds...or so it seems.

Most CGI scripts will run unmodified under mod_perl using the `Apache::Registryx` CGI compatability layer. But that's not the whole story. The exciting part is that mod_perl gives you access to the Apache API, letting you get at the innards of the Apache server and change its behavior in powerful and interesting ways. This section will give you a feel for the many things that you can do with mod_perl.

1.4.1 Creating Dynamic Pages

This is a ho-hum because you can do it with CGI and with `Apache::Registry`. Still, it's worth seeing a simple script written using the strict mod_perl API so you see what it looks like. Script III.1.1 prints out a little hello world message.

Install it by adding a section like this one to one of the configuration files:

```
<Location /hello/world>
  SetHandler  perl-script
  PerlHandler Apache::Hello
</Location>

Script III.1.1 Apache::Hello
----------------------------
package Apache::Hello;
# file: Apache/Hello.pm

use strict vars;
use Apache::Constants ':common';
```

```
sub handler {
    my $r = shift;
    $r->content_type('text/html');
    $r->send_http_header;
    my $host = $r->get_remote_host;
    $r->print(<<END);
<html>
<head>
<title>Hello There</title>
</head>
<body>
<h1>Hello $host</h1>
Hello to all the nice people at the Perl conference.  Lincoln is
trying really hard.  Be kind.
</body>
</html>
END
    return OK;
}
1;
```

You can do all the standard CGI stuff, such as reading the query string, creating fill-out forms, and so on. In fact, CGI.pm works with mod_perl, giving you the benefit of sticky forms, cookie handling, and elegant HTML generation.

1.4.2 File Filters

This is where the going gets fun. With mod_perl, you can install a *content handler* that works a lot like a four-letter word starrer-outer, but a lot faster.

1.4.2.1 Adding a Canned Footer to Every Page

Script III.2.1 adds a canned footer to every HTML file. The footer contains a copyright statement, plus the modification date of the file. You could easily extend this to add other information, such as a page hit counter, or the username of the page's owner.

This can be installed as the default handler for all files in a particular subdirectory like this:

```
<Location /footer>
  SetHandler perl-script
  PerlHandler Apache::Footer
</Location>
```

Or you can declare a new ".footer" extension and arrange for all files with this extension to be passed through the footer module:

```
AddType text/html .footer
<Files ~ "\.footer$">
    SetHandler  perl-script
    PerlHandler Apache::Footer
</Files>
```

```
Script III.2.1 Apache::Footer
-----------------------------
package Apache::Footer;
# file Apache::Footer.pm

use strict vars;
use Apache::Constants ':common';
use IO::File;

sub handler {
    my $r = shift;
    return DECLINED unless $r->content_type() eq 'text/html';
    my $file = $r->filename;
    return DECLINED unless $fh=IO::File->new($file);
    my $modtime = localtime((stat($file))[9]);
    my $footer=<<END;
<hr>
&copy; 1998 <a href="http://www.ora.com/">O\'Reilly &amp; Associates</a><br>
<em>Last Modified: $modtime</em>
END

    $r->send_http_header;

    while (<$fh>) {
        s!(</BODY>)!$footer$1!oi;
    } continue {
      $r->print($_);
    }

    return OK;
}

1;
```

For more customized footer/header handling, you might want to look at the `Apache::Sandwich` module on CPAN.

1.4.2.2 Dynamic Navigation Bar

Sick of hand-coding navigation bars in every HTML page? Less than enthused by the Java & JavaScript hacks? Here's a dynamic navigation bar implemented as a server side include.

First create a global configuration file for your site. The first column is the top of each major section. The second column is the label to print in the navigation bar

```
# Configuration file for the navigation bar
/index.html          Home
/new/                What's New
/tech/               Tech Support
/download/           Download
/dev/zero            Customer support
/dev/null            Complaints
```

Then, at the top (or bottom) of each HTML page that you want the navigation bar to appear on, add this comment:

```
<!--#NAVBAR-->
```

Now add `Apache::NavBar` to your system (Script III.2.2). This module parses the configuration file to create a "navigation bar object". We then call the navigation bar object's `to_html()` method in order to generate the HTML for the navigation bar to display on the current page (it will be different for each page, depending on what major section the page is in).

The next section does some checking to avoid transmitting the page again if it is already cached on the browser. The effective last modified time for the page is either the modification time of its HTML source code, or the navbar's configuration file modification date, whichever is more recent.

The remainder is just looping through the file a section at a time, searching for the `<!--NAVBAR-->` comment, and substituting the navigation bar HTML.

```
Script III.2.2 Apache::NavBar
-----------------------------

package Apache::NavBar;
# file Apache/NavBar.pm

use strict;
use Apache::Constants qw(:common);
use Apache::File ();

my %BARS = ();
my $TABLEATTS   = 'WIDTH="100%" BORDER=1';
my $TABLECOLOR  = '#C8FFFF';
my $ACTIVECOLOR = '#FF0000';

sub handler {
    my $r = shift;

    my $bar = read_configuration($r)         || return DECLINED;
    $r->content_type eq 'text/html'          || return DECLINED;
    my $fh = Apache::File->new($r->filename) || return DECLINED;
    my $navbar = $bar->to_html($r->uri);

    $r->update_mtime($bar->modified);
    $r->set_last_modified;
    my $rc = $r->meets_conditions;
    return $rc unless $rc == OK;

    $r->send_http_header;
    return OK if $r->header_only;

    local $/ = "";
    while (<$fh>) {
        s:<!--NAVBAR-->:$navbar:oi;
    } continue {
        $r->print($_);
```

```
    }

    return OK;
}


# read the navigation bar configuration file and return it as a
# hash.
sub read_configuration {
    my $r = shift;
    my $conf_file;
    return unless $conf_file = $r->dir_config('NavConf');
    return unless -e ($conf_file = $r->server_root_relative($conf_file));
    my $mod_time = (stat _)[9];
    return $BARS{$conf_file} if $BARS{$conf_file}
      && $BARS{$conf_file}->modified >= $mod_time;
    return $BARS{$conf_file} = NavBar->new($conf_file);
}

package NavBar;

# create a new NavBar object
sub new {
    my ($class,$conf_file) = @_;
    my (@c,%c);
    my $fh = Apache::File->new($conf_file) || return;
    while (<$fh>) {
        chomp;
        s/^\s+//; s/\s+$//;    #fold leading and trailing whitespace
        next if /^#/ || /^$/; # skip comments and empty lines
        next unless my($url, $label) = /^(\S+)\s+(.+)/;
        push @c, $url;      # keep the url in an ordered array
        $c{$url} = $label; # keep its label in a hash
    }
    return bless {'urls' => \@c,
                  'labels' => \%c,
                  'modified' => (stat $conf_file)[9]}, $class;
}

# return ordered list of all the URIs in the navigation bar
sub urls  { return @{shift->{'urls'}}; }

# return the label for a particular URI in the navigation bar
sub label { return $_[0]->{'labels'}->{$_[1]} || $_[1]; }

# return the modification date of the configuration file
sub modified { return $_[0]->{'modified'}; }

sub to_html {
    my $self = shift;
    my $current_url = shift;
    my @cells;
    for my $url ($self->urls) {
        my $label = $self->label($url);
        my $is_current = $current_url =~ /^$url/;
        my $cell = $is_current ?
            qq(<FONT COLOR="$ACTIVECOLOR">$label</FONT>)
```

```
                : qq(<A HREF="$url">$label</A>);
        push @cells,
        qq(<TD CLASS="navbar" ALIGN=CENTER BGCOLOR="$TABLECOLOR">$cell</TD>\n);
    }
    return qq(<TABLE $TABLEATTS><TR>@cells</TR></TABLE>\n);
}


1;
__END__

<Location />
  SetHandler  perl-script
  PerlHandler Apache::NavBar
  PerlSetVar  NavConf etc/navigation.conf
</Location>
```

`Apache::NavBar` is available on the CPAN, with further improvements.

1.4.2.3 On-the-Fly Compression

WU-FTP has a great feature that automatically gzips a file if you fetch it by name with a *.gz* extension added. Why can't Web servers do that trick? With Apache and mod_perl, you can.

Script III.2.4 is a content filter that automatically gzips everything retrieved from a particular directory and adds the "gzip" `Content-Encoding` header to it. Unix versions of Netscape Navigator will automatically recognize this encoding type and decompress the file on the fly. Windows and Mac versions don't. You'll have to save to disk and decompress, or install the WinZip plug-in. Bummer.

The code uses the `Compress::Zlib` module, and has to do a little fancy footwork (but not too much) to create the correct gzip header. You can extend this idea to do on-the-fly encryption, or whatever you like.

Here's the configuration entry you'll need. Everything in the */compressed* directory will be compressed automagically.

```
<Location /compressed>
   SetHandler  perl-script
   PerlHandler Apache::GZip
</Location>

Script III.2.3: Apache::GZip
--------------------------
package Apache::GZip;
#File: Apache::GZip.pm

use strict vars;
use Apache::Constants ':common';
use Compress::Zlib;
use IO::File;
use constant GZIP_MAGIC => 0x1f8b;
use constant OS_MAGIC => 0x03;

sub handler {
    my $r = shift;
```

```perl
    my ($fh,$gz);
    my $file = $r->filename;
    return DECLINED unless $fh=IO::File->new($file);
    $r->header_out('Content-Encoding'=>'gzip');
    $r->send_http_header;
    return OK if $r->header_only;

    tie *STDOUT,'Apache::GZip',$r;
    print($_) while <$fh>;
    untie *STDOUT;
    return OK;
}

sub TIEHANDLE {
    my($class,$r) = @_;
    # initialize a deflation stream
    my $d = deflateInit(-WindowBits=>-MAX_WBITS()) || return undef;

    # gzip header -- don't ask how I found out
    $r->print(pack("nccVcc",GZIP_MAGIC,Z_DEFLATED,0,time(),0,OS_MAGIC));

    return bless { r   => $r,
                   crc =>  crc32(undef),
                   d   => $d,
                   l   =>  0
                 },$class;

}

sub PRINT {
    my $self = shift;
    foreach (@_) {
      # deflate the data
      my $data = $self->{d}->deflate($_);
      $self->{r}->print($data);
      # keep track of its length and crc
      $self->{l} += length($_);
      $self->{crc} = crc32($_,$self->{crc});
    }
}

sub DESTROY {
   my $self = shift;

   # flush the output buffers
   my $data = $self->{d}->flush;
   $self->{r}->print($data);

   # print the CRC and the total length (uncompressed)
   $self->{r}->print(pack("LL",@{$self}{qw/crc l/}));
}

1;
```

For some alternatives that are being maintained, you might want to look at the `Apache::Compress` and `Apache::GzipChain` modules on CPAN, which can handle the output of any handler in a chain.

By adding a URI translation handler, you can set things up so that a remote user can append a *.gz* to the end of any URL and the file we be delivered in compressed form. Script III.2.4 shows the translation handler you need. It is called during the initial phases of the request to make any modifications to the URL that it wishes. In this case, it removes the *.gz* ending from the filename and arranges for `Apache:GZip` to be called as the content handler. The `lookup_uri()` call is used to exclude anything that has a special handler already defined (such as CGI scripts), and actual gzip files. The module replaces the information in the request object with information about the real file (without the `.gz`), and arranges for `Apache::GZip` to be the content handler for this file.

You just need this one directive to activate handling for all URLs at your site:

```
PerlTransHandler Apache::AutoGZip

Script III.2.4: Apache::AutoGZip
-----------------------------
package Apache::AutoGZip;

use strict 'vars';
use Apache::Constants qw/:common/;

sub handler {
    my $r = shift;

    # don't allow ourselves to be called recursively
    return DECLINED unless $r->is_initial_req;

    # don't do anything for files not ending with .gz
    my $uri = $r->uri;
    return DECLINED unless $uri=~/\.gz$/;
    my $basename = $`;

    # don't do anything special if the file actually exists
    return DECLINED if -e $r->lookup_uri($uri)->filename;

    # look up information about the file
    my $subr = $r->lookup_uri($basename);
    $r->uri($basename);
    $r->path_info($subr->path_info);
    $r->filename($subr->filename);

    # fix the handler to point to Apache::GZip;
    my $handler = $subr->handler;
    unless ($handler) {
      $r->handler('perl-script');
      $r->push_handlers('PerlHandler','Apache::GZip');
    } else {
      $r->handler($handler);
    }
```

```
    return OK;
}

1;
```

1.4.3 Access Control

Access control, as opposed to authentication and authorization, is based on something the user "is" rather than something he "knows". The "is" is usually something about his browser, such as its IP address, hostname, or user agent. Script III.3.1 blocks access to the Web server for certain User Agents (you might use this to block impolite robots).

`Apache::BlockAgent` reads its blocking information from a "bad agents" file, which contains a series of pattern matches. Most of the complexity of the code comes from watching this file and recompiling it when it changes. If the file doesn't change, it's is only read once and its patterns compiled in memory, making this module fast.

Here's an example bad agents file:

```
^teleport pro\/1\.28
^nicerspro
^mozilla\/3\.0 \(http engine\)
^netattache
^crescent internet toolpak http ole control v\.1\.0
^go-ahead-got-it
^wget
^devsoft's http component v1\.0
^www\.pl
^digout4uagent
```

A configuration entry to activate this blocker looks like this. In this case we're blocking access to the entire site. You could also block access to a portion of the site, or have different bad agents files associated with different portions of the document tree.

```
<Location />
  PerlAccessHandler Apache::BlockAgent
  PerlSetVar BlockAgentFile /home/www/conf/bad_agents.txt
</Location>

Script III.3.1: Apache::BlockAgent
-------------------------------
package Apache::BlockAgent;
# block browsers that we don't like

use strict 'vars';
use Apache::Constants ':common';
use IO::File;
my %MATCH_CACHE;
my $DEBUG = 0;

sub handler {
    my $r = shift;
```

```
        return DECLINED unless my $patfile = $r->dir_config('BlockAgentFile');
        return FORBIDDEN unless my $agent = $r->header_in('User-Agent');
        return SERVER_ERROR unless my $sub = get_match_sub($r,$patfile);
        return OK if $sub->($agent);
        $r->log_reason("Access forbidden to agent $agent",$r->filename);
        return FORBIDDEN;
    }

    # This routine creates a pattern matching subroutine from a
    # list of pattern matches stored in a file.
    sub get_match_sub {
        my ($r,$filename) = @_;
        my $mtime = -M $filename;

        # try to return the sub from cache
        return $MATCH_CACHE{$filename}->{'sub'} if
            $MATCH_CACHE{$filename} &&
                $MATCH_CACHE{$filename}->{'mod'} <= $mtime;

        # if we get here, then we need to create the sub
        return undef unless my $fh = new IO::File($filename);
        chomp(my @pats = <$fh>); # get the patterns into an array
        my $code = "sub { \$_ = shift;\n";
        foreach (@pats) {
            next if /^#/
            $code .= "return undef if /$_/i;\n";

        }
        $code .= "1; }\n";
        warn $code if $DEBUG;

        # create the sub, cache and return it
        my $sub = eval $code;
        unless ($sub) {
            $r->log_error($r->uri,": ",$@);
            return undef;
        }
        @{$MATCH_CACHE{$filename}}{'sub','mod'}=($sub,$modtime);
        return $MATCH_CACHE{$filename}->{'sub'};
    }

    1;
```

1.4.4 Authentication and Authorization

Thought you were stuck with authentication using text, DBI and DBM files? mod_perl opens the authentication/authorization API wide. The two phases are authentication, in which the user has to prove who he or she is (usually by providing a username and password), and authorization, in which the system decides whether this user has sufficient privileges to view the requested URL. A scheme can incorporate authentication and authorization either together or singly.

1.4.4.1 Authentication with NIS

If you keep Unix system passwords in */etc/passwd* or distribute them by NIS (not NIS+) you can authenticate Web users against the system password database. (It's not a good idea to do this if the system is connected to the Internet because passwords travel in the clear, but it's OK for trusted intranets.)

Script III.4.1 shows how the `Apache::AuthSystem` module fetches the user's name and password, compares it to the system password, and takes appropriate action. The `getpwnam()` function operates either on local files or on the NIS database, depending on how the server host is configured. WARNING: the module will fail if you use a shadow password system, since the Web server doesn't have root privileges.

In order to activate this system, put a configuration directive like this one in access.conf:

```
<Location /protected>
  AuthName Test
  AuthType Basic
  PerlAuthenHandler Apache::AuthSystem
  require valid-user
</Location>

Script III.4.1: Apache::AuthSystem
-------------------------------
package Apache::AuthSystem;
# authenticate users on system password database

use strict;
use Apache::Constants ':common';

sub handler {
    my $r = shift;

    my($res, $sent_pwd) = $r->get_basic_auth_pw;
    return $res if $res != OK;

    my $user = $r->connection->user;
    my $reason = "";

    my($name,$passwd) = getpwnam($user);
    if (!$name) {
        $reason = "user does not have an account on this system";
    } else {
        $reason = "user did not provide correct password"
          unless $passwd eq crypt($sent_pwd,$passwd);
    }

    if($reason) {
        $r->note_basic_auth_failure;
        $r->log_reason($reason,$r->filename);
        return AUTH_REQUIRED;
    }
```

```
    return OK;
}

1;
```

There are modules doing equivalent things on CPAN: `Apache::AuthenPasswd` and `Apache::AuthxPasswd`.

1.4.4.2 Anonymous Authentication

Here's a system that authenticates users the way anonymous FTP does. They have to enter a name like "Anonymous" (configurable) and a password that looks like a valid e-mail address. The system rejects the username and password unless they are formatted correctly.

In a real application, you'd probably want to log the password somewhere for posterity. Script III.4.2 shows the code for `Apache::AuthAnon`. To activate it, create a *httpd.conf* section like this one:

```
<Location /protected>
      AuthName Anonymous
      AuthType Basic
      PerlAuthenHandler Apache::AuthAnon
      require valid-user

      PerlSetVar Anonymous anonymous|anybody
</Location>

Script III.4.2: Anonymous Authentication
---------------------------------------
package Apache::AuthAnon;

use strict;
use Apache::Constants ':common';

my $email_pat = '\w+\@\w+\.\w+';
my $anon_id  = "anonymous";

sub handler {
    my $r = shift;

    my($res, $sent_pwd) = $r->get_basic_auth_pw;
    return $res if $res != OK;

    my $user = lc $r->connection->user;
    my $reason = "";

    my $check_id = $r->dir_config("Anonymous") || $anon_id;

    unless($user =~ /^$check_id$/i) {
      $reason = "user did not enter a valid anonymous username";
    }

    unless($sent_pwd =~ /$email_pat/o) {
      $reason = "user did not enter an email address password";
    }
```

```
     if($reason) {
       $r->note_basic_auth_failure;
       $r->log_reason($reason,$r->filename);
       return AUTH_REQUIRED;
     }

     $r->notes(AuthAnonPassword => $sent_pwd);

     return OK;
}

1;
```

1.4.4.3 Gender-Based Authorization

After authenticating, you can authorize. The most familiar type of authorization checks a group database to see if the user belongs to one or more privileged groups. But authorization can be anything you dream up.

Script III.4.3 shows how you can authorize users by their gender (or at least their *apparent* gender, by checking their names with Jon Orwant's `Text::GenderFromName` module. This must be used in conjunction with an authentication module, such as one of the standard Apache modules or a custom one.

This configuration restricts access to users with feminine names, except for the users "Webmaster" and "Jeff", who are allowed access.

```
<Location /ladies_only>
  AuthName "Ladies Only"
  AuthType Basic
  AuthUserFile /home/www/conf/users.passwd
  PerlAuthzHandler  Apache::AuthzGender
  require gender F           # allow females
  require user Webmaster Jeff # allow Webmaster or Jeff
</Location>
```

The script uses a custom error response to explain why the user was denied admittance. This is better than the standard "Authorization Failed" message.

```
Script III.4.3: Apache::AuthzGender
-------------------------------
package Apache::AuthzGender;

use strict;
use Text::GenderFromName;
use Apache::Constants ":common";

my %G=('M'=>"male",'F'=>"female");

sub handler {
    my $r = shift;

    return DECLINED unless my $requires = $r->requires;
    my $user = lc($r->connection->user);
    substr($user,0,1)=~tr/a-z/A-Z/;
```

```
    my $guessed_gender = uc(gender($user)) || 'M';

    my $explanation = <<END;
<html><head><title>Unauthorized</title></head><body>
<h1>You Are Not Authorized to Access This Page</h1>
Access to this page is limited to:
<ol>
END

    foreach (@$requires) {
      my ($requirement,@rest ) = split(/\s+/,$_->{requirement});
      if (lc $requirement eq 'user') {
          foreach (@rest) { return OK if $user eq $_; }
          $explanation .= "<LI>Users @rest.\n";
      } elsif (lc $requirement eq 'gender') {
          foreach (@rest) { return OK if $guessed_gender eq uc $_; }
          $explanation .= "<LI>People of the @G{@rest} persuasion.\n";
      } elsif (lc $requirement eq 'valid-user') {
          return OK;
      }
    }

    $explanation .= "</OL></BODY></HTML>";

    $r->custom_response(AUTH_REQUIRED,$explanation);
    $r->note_basic_auth_failure;
    $r->log_reason("user $user: not authorized",$r->filename);
    return AUTH_REQUIRED;
}

1;
```

`Apache::AuthzGender` is available from the CPAN.

1.4.5 Proxy Services

mod_perl gives you access to Apache's ability to act as a Web proxy. You can intervene at any step in the proxy transaction to modify the outgoing request (for example, stripping off headers in order to create an anonymizing proxy) or to modify the returned page.

1.4.5.1 A Banner Ad Blocker

Script III.5.1 shows the code for a banner-ad blocker written by Doug MacEachern. It intercepts all proxy requests, substituting its own content handler for the default. The content handler uses the LWP library to fetch the requested document. If the retrieved document is an image, and its URL matches the pattern (ads?|advertisement|banner), then the content of the image is replaced with a dynamically-generated GIF that reads "Blocked Ad". The generated image is exactly the same size as the original, preserving the page layout. Notice how the outgoing headers from the Apache request object are copied to the LWP request, and how the incoming LWP response headers are copied back to Apache. This makes the transaction nearly transparent to Apache and to the remote server.

In addition to LWP you'll need `GD.pm` and `Image::Size` to run this module. To activate it, add the following line to the configuration file:

```
PerlTransHandler Apache::AdBlocker
```

Then configure your browser to use the server to proxy all its HTTP requests. Works like a charm! With a little more work, and some help from the `ImageMagick` module, you could adapt this module to quiet-down animated GIFs by stripping them of all but the very first frame.

```
Script III.5.1: Apache::AdBlocker
-------------------------------

package Apache::AdBlocker;

use strict;
use vars qw(@ISA $VERSION);
use Apache::Constants qw(:common);
use GD ();
use Image::Size qw(imgsize);
use LWP::UserAgent ();

@ISA = qw(LWP::UserAgent);
$VERSION = '1.00';

my $UA = __PACKAGE__->new;
$UA->agent(join "/", __PACKAGE__, $VERSION);

my $Ad = join "|", qw{ads? advertisement banner};

sub handler {
    my($r) = @_;
    return DECLINED unless $r->proxyreq;
    $r->handler("perl-script"); #ok, let's do it
    $r->push_handlers(PerlHandler => \&proxy_handler);
    return OK;
}

sub proxy_handler {
    my($r) = @_;

    my $request = HTTP::Request->new($r->method, $r->uri);

    $r->headers_in->do(sub {
       $request->header(@_);
    });

    # copy POST data, if any
    if($r->method eq 'POST') {
       my $len = $r->header_in('Content-length');

       my $buf;
       $r->read($buf, $len);
       $request->content($buf);
       $request->content_type($r->content_type);
    }
```

```
    my $response = $UA->request($request);
    $r->content_type($response->header('Content-type'));

    #feed response back into our request_rec*
    $r->status($response->code);
    $r->status_line(join " ", $response->code, $response->message);
    $response->scan(sub {
        $r->header_out(@_);
    });

    if ($r->header_only) {
        $r->send_http_header();
        return OK;
    }

    my $content = \$response->content;
    if($r->content_type =~ /^image/ and $r->uri =~ /\b($Ad)\b/i) {
        block_ad($content);
        $r->content_type("image/gif");
    }

    $r->content_type('text/html') unless $$content;
    $r->send_http_header;
    $r->print($$content || $response->error_as_HTML);

    return OK;
}

sub block_ad {
    my $data = shift;
    my($x, $y) = imgsize($data);

    my $im = GD::Image->new($x,$y);

    my $white = $im->colorAllocate(255,255,255);
    my $black = $im->colorAllocate(0,0,0);
    my $red = $im->colorAllocate(255,0,0);

    $im->transparent($white);
    $im->string(GD::gdLargeFont(),5,5,"Blocked Ad",$red);
    $im->rectangle(0,0,$x-1,$y-1,$black);

    $$data = $im->gif;
}

1;
```

Another way of doing this module would be to scan all proxied HTML files for <img> tags containing one of the verboten URLs, then replacing the src attribute with a transparent GIF of our own. However, unless the <img> tag contained width and height attributes, we wouldn't be able to return a GIF of the correct size -- unless we were to go hunting for the GIF with LWP, in which case we might as well do it this way.

1.4.6 Customized Logging

After Apache handles a transaction, it passes all the information about the transaction to the log handler. The default log handler writes out lines to the log file. With mod_perl, you can install your own log handler to do customized logging.

1.4.6.1 Send E-Mail When a Particular Page Gets Hit

Script III.6.1 installs a log handler which watches over a page or set of pages. When someone fetches a watched page, the log handler sends off an e-mail to notify someone (probably the owner of the page) that the page has been read.

To activate the module, just attach a `PerlLogHandler` to the `<Location>` or `<Files>` you wish to watch. For example:

```
  <Location /~lstein>
     PerlLogHandler Apache::LogMail
     PerlSetVar mailto lstein@cshl.org
  </Location>
```

The "mailto" directive specifies the name of the recipient(s) to notify.

```
  Script III.6.1: Apache::LogMail
  -----------------------------
 package Apache::LogMail;
 use Apache::Constants ':common';

 sub handler {
     my $r = shift;
     my $mailto = $r->dir_config('mailto');
     return DECLINED unless $mailto
     my $request = $r->the_request;
     my $uri = $r->uri;
     my $agent = $r->header_in("User-agent");
     my $bytes = $r->bytes_sent;
     my $remote = $r->get_remote_host;
     my $status = $r->status_line;
     my $date = localtime;
     unless (open (MAIL,"|/usr/lib/sendmail -oi -t")) {
        $r->log_error("Couldn't open mail: $!");
        return DECLINED;
     }
     print MAIL <<END;
 To: $mailto
 From: Mod Perl <webmaster>
 Subject: Somebody looked at $uri

 At $date, a user at $remote looked at
 $uri using the $agent browser.

 The request was $request,
 which resulted returned a code of $status.

 $bytes bytes were transferred.
```

```
END
    close MAIL;
    return OK;
}
1;
```

1.4.6.2 Writing Log Information Into a Relational Database

Coming full circle, Script III.6.2 shows a module that writes log information into a DBI database. The idea is similar to Script I.1.9, but there's now no need to open a pipe to an external process. It's also a little more efficient, because the log data fields can be recovered directly from the Apache request object, rather than parsed out of a line of text. Another improvement is that we can set up the Apache configuration files so that only accesses to certain directories are logged in this way.

To activate, add something like this to your configuration file: PerlLogHandler Apache::LogDBI

Or, to restrict special logging to accesses of files in below the URL "/lincoln_logs" add this:

```
<Location /lincoln_logs>
  PerlLogHandler Apache::LogDBI
</Location>

Script III.6.2: Apache::LogDBI
------------------------------
package Apache::LogDBI;
use Apache::Constants ':common';

use strict 'vars';
use vars qw($DB $STH);
use DBI;
use POSIX 'strftime';

use constant DSN       => 'dbi:mysql:www';
use constant DB_TABLE  => 'access_log';
use constant DB_USER   => 'nobody';
use constant DB_PASSWD => '';

$DB = DBI->connect(DSN,DB_USER,DB_PASSWD) || die DBI->errstr;
$STH = $DB->prepare("INSERT INTO ${\DB_TABLE} VALUES(?,?,?,?,?,?,?,?,?)")
      || die $DB->errstr;

sub handler {
    my $r = shift;
    my $date    = strftime('%Y-%m-%d %H:%M:%S',localtime);
    my $host    = $r->get_remote_host;
    my $method  = $r->method;
    my $url     = $r->uri;
    my $user    = $r->connection->user;
    my $referer = $r->header_in('Referer');
    my $browser = $r->header_in("User-agent");
    my $status  = $r->status;
    my $bytes   = $r->bytes_sent;
    $STH->execute($date,$host,$method,$url,$user,
                  $browser,$referer,$status,$bytes);
```

```
    return OK;
}

1;
```

There are other alternatives which are more actively maintained available from the CPAN: `Apache::DBILogger` and `Apache::DBILogConfig`.

1.5 Conclusion

These tricks illustrate the true power of mod_perl; not only were Perl and Apache good friends from the start, thanks to Perl's excellent text-handling capacity, but when mod_perl is used, your complete access to the Apache API gives you unprecendented power in dynamic web serving.

To find more tips and tricks, look for modules on the CPAN, look through the mod_perl documentation, and also in the following books by Lincoln Stein:

- **"How to Set Up and Maintain a Web Site"**

  General introduction to Web site care and feeding, with an emphasis on Apache. Addison-Wesley 1997.

  Companion Web site at http://www.genome.wi.mit.edu/WWW/

- **"Web Security, a Step-by-Step Reference Guide"**

  How to keep your Web site free from thieves, vandals, hooligans and other yahoos. Addison-Wesley 1998.

  Companion Web site at http://www.w3.org/Security/Faq/

- **"The Official Guide to Programming with CGI.pm"**

  Everything I know about CGI.pm (and some things I don't!). John Wiley & Sons, 1998.

  Companion Web site at http://www.wiley.com/legacy/compbooks/stein/

- **"Writing Apache Modules in Perl and C"**

  Co-authored with Doug MacEachern. O'Reilly & Associates.

  Companion Web site at http://www.modperl.com/

- **WebTechniques Columns**

  I write a monthly column for WebTechniques magazine (now New Architect). You can find back-issues and reprints at http://www.webtechniques.com/

- **The Perl Journal Columns**

  I write a quarterly column for TPJ. Source code listings are available at http://www.tpj.com/

1.6 Maintainers

The maintainer is the person(s) you should contact with updates, corrections and patches.

- Per Einar Ellefsen <per.einar (at) skynet.be>

1.7 Authors

- **Lincoln Stein <lstein (at) cshl.org>**

Only the major authors are listed above. For contributors see the Changes file.

Table of Contents:

Table of Contents: