NAME

perlfaq9 - Networking

DESCRIPTION

This section deals with questions related to networking, the internet, and a few on the web.

What is the correct form of response from a CGI script?

(Alan Flavell <flavell+www@a5.ph.gla.ac.uk> answers...)

The Common Gateway Interface (CGI) specifies a software interface between a program ("CGI script") and a web server (HTTPD). It is not specific to Perl, and has its own FAQs and tutorials, and usenet group, comp.infosystems.www.authoring.cgi

The CGI specification is outlined in an informational RFC: http://www.ietf.org/rfc/rfc3875

These Perl FAQs very selectively cover some CGI issues. However, Perl programmers are strongly advised to use the `CGI.pm` module, to take care of the details for them.

The similarity between CGI response headers (defined in the CGI specification) and HTTP response headers (defined in the HTTP specification, RFC2616) is intentional, but can sometimes be confusing.

The CGI specification defines two kinds of script: the "Parsed Header" script, and the "Non Parsed Header" (NPH) script. Check your server documentation to see what it supports. "Parsed Header" scripts are simpler in various respects. The CGI specification allows any of the usual newline representations in the CGI response (it's the server's job to create an accurate HTTP response based on it). So "\n" written in text mode is technically correct, and recommended. NPH scripts are more tricky: they must put out a complete and accurate set of HTTP transaction response headers; the HTTP specification calls for records to be terminated with carriage-return and line-feed; i.e., ASCII \015\012 written in binary mode.

Using `CGI.pm` gives excellent platform independence, including EBCDIC systems. `CGI.pm` selects an appropriate newline representation (`$CGI::CRLF`) and sets binmode as appropriate.

My CGI script runs from the command line but not the browser. (500 Server Error)

(contributed by brian d foy)

There are many things that might be wrong with your CGI program, and only some of them might be related to Perl. Try going through the troubleshooting guide on Perlmonks:

```
http://www.perlmonks.org/?node_id=380424
```

How can I get better error messages from a CGI program?

Use the `CGI::Carp` module. It replaces `warn` and `die`, plus the normal `Carp` module's `carp`, `croak`, and `confess` functions with more verbose and safer versions. It still sends them to the normal server error log.

```
use CGI::Carp;
warn "This is a complaint";
die "But this one is serious";
```

The following use of `CGI::Carp` also redirects errors to a file of your choice, placed in a `BEGIN` block to catch compile-time warnings as well:

```
BEGIN {
 use CGI::Carp qw(carpout);
 open(LOG, ">>/var/local/cgi-logs/mycgi-log")
  or die "Unable to append to mycgi-log: $!\n";
 carpout(*LOG);
```

```
}
```

You can even arrange for fatal errors to go back to the client browser, which is nice for your own debugging, but might confuse the end user.

```
use CGI::Carp qw(fatalsToBrowser);
die "Bad error here";
```

Even if the error happens before you get the HTTP header out, the module will try to take care of this to avoid the dreaded server 500 errors. Normal warnings still go out to the server error log (or wherever you've sent them with `carpout`) with the application name and date stamp prepended.

How do I remove HTML from a string?

The most correct way (albeit not the fastest) is to use `HTML::Parser` from CPAN. Another mostly correct way is to use `HTML::FormatText` which not only removes HTML but also attempts to do a little simple formatting of the resulting plain text.

Many folks attempt a simple-minded regular expression approach, like `s/<.*?>//g`, but that fails in many cases because the tags may continue over line breaks, they may contain quoted angle-brackets, or HTML comments may be present. Plus, folks forget to convert entities--like `<` for example.

Here's one "simple-minded" approach, that works for most files:

```
#!/usr/bin/perl -p0777
s/<(?:[^>'"]*|(['"]).*?\g1)*>//gs
```

If you want a more complete solution, see the 3-stage striphtml program in http://www.cpan.org/authors/Tom_Christiansen/scripts/striphtml.gz .

Here are some tricky cases that you should think about when picking a solution:

```
<IMG SRC = "foo.gif" ALT = "A > B">

<IMG SRC = "foo.gif"
 ALT = "A > B">

<!-- <A comment> -->

<script>if (a<b && a>c)</script>

<# Just data #>

<![INCLUDE CDATA [ >>>>>>>>>>>> ]]>
```

If HTML comments include other tags, those solutions would also break on text like this:

```
<!-- This section commented out.
 <B>You can't see me!</B>
-->
```

How do I extract URLs?

You can easily extract all sorts of URLs from HTML with `HTML::SimpleLinkExtor` which handles anchors, images, objects, frames, and many other tags that can contain a URL. If you need anything more complex, you can create your own subclass of `HTML::LinkExtor` or `HTML::Parser`. You

might even use HTML::SimpleLinkExtor as an example for something specifically suited to your needs.

You can use URI::Find to extract URLs from an arbitrary text document.

Less complete solutions involving regular expressions can save you a lot of processing time if you know that the input is simple. One solution from Tom Christiansen runs 100 times faster than most module-based approaches but only extracts URLs from anchors where the first attribute is HREF and there are no other attributes.

```
#!/usr/bin/perl -n00
# qxurl - tchrist@perl.com
print "$2\n" while m{
 < \s*
   A \s+ HREF \s* = \s* (["']) (.*?) \g1
 \s* >
}gsix;
```

How do I download a file from the user's machine? How do I open a file on another machine?

In this case, download means to use the file upload feature of HTML forms. You allow the web surfer to specify a file to send to your web server. To you it looks like a download, and to the user it looks like an upload. No matter what you call it, you do it with what's known as **multipart/form-data** encoding. The CGI.pm module (which comes with Perl as part of the Standard Library) supports this in the start_multipart_form() method, which isn't the same as the startform() method.

See the section in the CGI.pm documentation on file uploads for code examples and details.

How do I make an HTML pop-up menu with Perl?

(contributed by brian d foy)

The CGI.pm module (which comes with Perl) has functions to create the HTML form widgets. See the CGI.pm documentation for more examples.

```
use CGI qw/:standard/;
print header,
 start_html('Favorite Animals'),

 start_form,
  "What's your favorite animal? ",
 popup_menu(
  -name   => 'animal',
  -values => [ qw( Llama Alpaca Camel Ram ) ]
  ),
 submit,

 end_form,
 end_html;
```

How do I fetch an HTML file?

(contributed by brian d foy)

Use the libwww-perl distribution. The LWP::Simple module can fetch web resources and give their content back to you as a string:

```
use LWP::Simple qw(get);
```

```
my $html = get( "http://www.example.com/index.html" );
```

It can also store the resource directly in a file:

```
use LWP::Simple qw(getstore);

getstore( "http://www.example.com/index.html", "foo.html" );
```

If you need to do something more complicated, you can use `LWP::UserAgent` module to create your own user-agent (e.g. browser) to get the job done. If you want to simulate an interactive web browser, you can use the `WWW::Mechanize` module.

How do I automate an HTML form submission?

If you are doing something complex, such as moving through many pages and forms or a web site, you can use `WWW::Mechanize`. See its documentation for all the details.

If you're submitting values using the GET method, create a URL and encode the form using the `query_form` method:

```
use LWP::Simple;
use URI::URL;

my $url = url('http://www.perl.com/cgi-bin/cpan_mod');
$url->query_form(module => 'DB_File', readme => 1);
$content = get($url);
```

If you're using the POST method, create your own user agent and encode the content appropriately.

```
use HTTP::Request::Common qw(POST);
use LWP::UserAgent;

$ua = LWP::UserAgent->new();
my $req = POST 'http://www.perl.com/cgi-bin/cpan_mod',
      [ module => 'DB_File', readme => 1 ];
$content = $ua->request($req)->as_string;
```

How do I decode or create those %-encodings on the web?

(contributed by brian d foy)

Those `%` encodings handle reserved characters in URIs, as described in RFC 2396, Section 2. This encoding replaces the reserved character with the hexadecimal representation of the character's number from the US-ASCII table. For instance, a colon, `:`, becomes `%3A`.

In CGI scripts, you don't have to worry about decoding URIs if you are using `CGI.pm`. You shouldn't have to process the URI yourself, either on the way in or the way out.

If you have to encode a string yourself, remember that you should never try to encode an already-composed URI. You need to escape the components separately then put them together. To encode a string, you can use the `URI::Escape` module. The `uri_escape` function returns the escaped string:

```
my $original = "Colon : Hash # Percent %";

my $escaped = uri_escape( $original );
```

```
print "$escaped\n"; # 'Colon%20%3A%20Hash%20%23%20Percent%20%25'
```

To decode the string, use the `uri_unescape` function:

```
my $unescaped = uri_unescape( $escaped );

print $unescaped; # back to original
```

If you wanted to do it yourself, you simply need to replace the reserved characters with their encodings. A global substitution is one way to do it:

```
# encode
$string =~ s/([^^A-Za-z0-9\-_.!~*'()])/ sprintf "%%%0x", ord $1 /eg;

#decode
$string =~ s/%([A-Fa-f\d]{2})/chr hex $1/eg;
```

How do I redirect to another page?

Specify the complete URL of the destination (even if it is on the same server). This is one of the two different kinds of CGI "Location:" responses which are defined in the CGI specification for a Parsed Headers script. The other kind (an absolute URLpath) is resolved internally to the server without any HTTP redirection. The CGI specifications do not allow relative URLs in either case.

Use of `CGI.pm` is strongly recommended. This example shows redirection with a complete URL. This redirection is handled by the web browser.

```
use CGI qw/:standard/;

my $url = 'http://www.cpan.org/';
print redirect($url);
```

This example shows a redirection with an absolute URLpath. This redirection is handled by the local web server.

```
my $url = '/CPAN/index.html';
print redirect($url);
```

But if coded directly, it could be as follows (the final "\n" is shown separately, for clarity), using either a complete URL or an absolute URLpath.

```
print "Location: $url\n";   # CGI response header
print "\n";                 # end of headers
```

How do I put a password on my web pages?

To enable authentication for your web server, you need to configure your web server. The configuration is different for different sorts of web servers--apache does it differently from iPlanet which does it differently from IIS. Check your web server documentation for the details for your particular server.

How do I edit my .htpasswd and .htgroup files with Perl?

The `HTTPD::UserAdmin` and `HTTPD::GroupAdmin` modules provide a consistent OO interface to these files, regardless of how they're stored. Databases may be text, dbm, Berkeley DB or any database with a DBI compatible driver. `HTTPD::UserAdmin` supports files used by the "Basic" and "Digest" authentication schemes. Here's an example:

```
use HTTPD::UserAdmin ();
HTTPD::UserAdmin
  ->new(DB => "/foo/.htpasswd")
  ->add($username => $password);
```

How do I make sure users can't enter values into a form that cause my CGI script to do bad things?

(contributed by brian d foy)

You can't prevent people from sending your script bad data. Even if you add some client-side checks, people may disable them or bypass them completely. For instance, someone might use a module such as `LWP` to access your CGI program. If you want to prevent data that try to use SQL injection or other sorts of attacks (and you should want to), you have to not trust any data that enter your program.

The *perlsec* documentation has general advice about data security. If you are using the `DBI` module, use placeholder to fill in data. If you are running external programs with `system` or `exec`, use the list forms. There are many other precautions that you should take, too many to list here, and most of them fall under the category of not using any data that you don't intend to use. Trust no one.

How do I parse a mail header?

For a quick-and-dirty solution, try this solution derived from *"split" in perlfunc*:

```
$/ = '';
$header = <MSG>;
$header =~ s/\n\s+/ /g;  # merge continuation lines
%head = ( UNIX_FROM_LINE, split /^([-\w]+):\s*/m, $header );
```

That solution doesn't do well if, for example, you're trying to maintain all the Received lines. A more complete approach is to use the `Mail::Header` module from CPAN (part of the `MailTools` package).

How do I decode a CGI form?

(contributed by brian d foy)

Use the `CGI.pm` module that comes with Perl. It's quick, it's easy, and it actually does quite a bit of work to ensure things happen correctly. It handles GET, POST, and HEAD requests, multipart forms, multivalued fields, query string and message body combinations, and many other things you probably don't want to think about.

It doesn't get much easier: the `CGI.pm` module automatically parses the input and makes each value available through the `param()` function.

```
use CGI qw(:standard);

my $total = param( 'price' ) + param( 'shipping' );

my @items = param( 'item' ); # multiple values, same field name
```

If you want an object-oriented approach, `CGI.pm` can do that too.

```
use CGI;

my $cgi = CGI->new();

my $total = $cgi->param( 'price' ) + $cgi->param( 'shipping' );
```

```
my @items = $cgi->param( 'item' );
```

You might also try `CGI::Minimal` which is a lightweight version of the same thing. Other CGI::* modules on CPAN might work better for you, too.

Many people try to write their own decoder (or copy one from another program) and then run into one of the many "gotchas" of the task. It's much easier and less hassle to use `CGI.pm`.

How do I check a valid mail address?

(partly contributed by Aaron Sherman)

This isn't as simple a question as it sounds. There are two parts:

a) How do I verify that an email address is correctly formatted?

b) How do I verify that an email address targets a valid recipient?

Without sending mail to the address and seeing whether there's a human on the other end to answer you, you cannot fully answer part *b*, but either the `Email::Valid` or the `RFC::RFC822::Address` module will do both part *a* and part *b* as far as you can in real-time.

If you want to just check part *a* to see that the address is valid according to the mail header standard with a simple regular expression, you can have problems, because there are deliverable addresses that aren't RFC-2822 (the latest mail header standard) compliant, and addresses that aren't deliverable which, are compliant. However, the following will match valid RFC-2822 addresses that do not have comments, folding whitespace, or any other obsolete or non-essential elements. This *just* matches the address itself:

```
my $atom       = qr{[a-zA-Z0-9_!#\$\%&'*+/=?\^`{}~|\-]+};
my $dot_atom   = qr{$atom(?:\.$atom)*};
my $quoted     = qr{"(?:\\[^\r\n]|[^\\"])*"};
my $local      = qr{(?:$dot_atom|$quoted)};
my $quotedpair = qr{\\[\x00-\x09\x0B-\x0c\x0e-\x7e]};
my $domain_lit = qr{\[(?:$quotedpair|[\x21-\x5a\x5e-\x7e])*\]};
 my $domain     = qr{(?:$dot_atom|$domain_lit)};
my $addr_spec  = qr{$local\@$domain};
```

Just match an address against `/^${addr_spec}$/` to see if it follows the RFC2822 specification. However, because it is impossible to be sure that such a correctly formed address is actually the correct way to reach a particular person or even has a mailbox associated with it, you must be very careful about how you use this.

Our best advice for verifying a person's mail address is to have them enter their address twice, just as you normally do to change a password. This usually weeds out typos. If both versions match, send mail to that address with a personal message. If you get the message back and they've followed your directions, you can be reasonably assured that it's real.

A related strategy that's less open to forgery is to give them a PIN (personal ID number). Record the address and PIN (best that it be a random one) for later processing. In the mail you send, ask them to include the PIN in their reply. But if it bounces, or the message is included via a "vacation" script, it'll be there anyway. So it's best to ask them to mail back a slight alteration of the PIN, such as with the characters reversed, one added or subtracted to each digit, etc.

How do I decode a MIME/BASE64 string?

The `MIME-Base64` package (available from CPAN) handles this as well as the MIME/QP encoding. Decoding BASE64 becomes as simple as:

```
use MIME::Base64;
$decoded = decode_base64($encoded);
```

The MIME-Tools package (available from CPAN) supports extraction with decoding of BASE64 encoded attachments and content directly from email messages.

If the string to decode is short (less than 84 bytes long) a more direct approach is to use the unpack() function's "u" format after minor transliterations:

```
tr#A-Za-z0-9+/##cd;                  # remove non-base64 chars
tr#A-Za-z0-9+/# -_#;                 # convert to uuencoded format
$len = pack("c", 32 + 0.75*length);  # compute length byte
print unpack("u", $len . $_);        # uudecode and print
```

How do I return the user's mail address?

On systems that support getpwuid, the $< variable, and the Sys::Hostname module (which is part of the standard perl distribution), you can probably try using something like this:

```
use Sys::Hostname;
$address = sprintf('%s@%s', scalar getpwuid($<), hostname);
```

Company policies on mail address can mean that this generates addresses that the company's mail system will not accept, so you should ask for users' mail addresses when this matters. Furthermore, not all systems on which Perl runs are so forthcoming with this information as is Unix.

The Mail::Util module from CPAN (part of the MailTools package) provides a mailaddress() function that tries to guess the mail address of the user. It makes a more intelligent guess than the code above, using information given when the module was installed, but it could still be incorrect. Again, the best way is often just to ask the user.

How do I send mail?

Use the sendmail program directly:

```
open(SENDMAIL, "|/usr/lib/sendmail -oi -t -odq")
 or die "Can't fork for sendmail: $!\n";
print SENDMAIL <<"EOF";
From: User Originating Mail <me\@host>
To: Final Destination <you\@otherhost>
Subject: A relevant subject line

Body of the message goes here after the blank line
in as many lines as you like.
EOF
close(SENDMAIL)     or warn "sendmail didn't close nicely";
```

The **-oi** option prevents sendmail from interpreting a line consisting of a single dot as "end of message". The **-t** option says to use the headers to decide who to send the message to, and **-odq** says to put the message into the queue. This last option means your message won't be immediately delivered, so leave it out if you want immediate delivery.

Alternate, less convenient approaches include calling mail (sometimes called mailx) directly or simply opening up port 25 have having an intimate conversation between just you and the remote SMTP daemon, probably sendmail.

Or you might be able use the CPAN module Mail::Mailer:

```
use Mail::Mailer;

$mailer = Mail::Mailer->new();
$mailer->open({ From    => $from_address,
```

```
    To      => $to_address,
    Subject => $subject,
     })
 or die "Can't open: $!\n";
print $mailer $body;
$mailer->close();
```

The `Mail::Internet` module uses `Net::SMTP` which is less Unix-centric than `Mail::Mailer`, but less reliable. Avoid raw SMTP commands. There are many reasons to use a mail transport agent like `sendmail`. These include queuing, MX records, and security.

How do I use MIME to make an attachment to a mail message?

This answer is extracted directly from the `MIME::Lite` documentation. Create a multipart message (i.e., one with attachments).

```
use MIME::Lite;

### Create a new multipart message:
$msg = MIME::Lite->new(
    From    =>'me@myhost.com',
    To      =>'you@yourhost.com',
    Cc      =>'some@other.com, some@more.com',
    Subject =>'A message with 2 parts...',
    Type    =>'multipart/mixed'
    );

### Add parts (each "attach" has same arguments as "new"):
$msg->attach(Type     =>'TEXT',
    Data     =>"Here's the GIF file you wanted"
    );
$msg->attach(Type     =>'image/gif',
    Path     =>'aaa000123.gif',
    Filename =>'logo.gif'
    );

$text = $msg->as_string;
```

`MIME::Lite` also includes a method for sending these things.

```
$msg->send;
```

This defaults to using *sendmail* but can be customized to use SMTP via *Net::SMTP*.

How do I read mail?

While you could use the `Mail::Folder` module from CPAN (part of the `MailFolder` package) or the `Mail::Internet` module from CPAN (part of the `MailTools` package), often a module is overkill. Here's a mail sorter.

```
#!/usr/bin/perl

my(@msgs, @sub);
my $msgno = -1;
$/ = '';                    # paragraph reads
while (<>) {
 if (/^From /m) {
```

```
  /^Subject:\s*(?:Re:\s*)*(.*)/mi;
  $sub[++$msgno] = lc($1) || '';
 }
 $msgs[$msgno] .= $_;
}
for my $i (sort { $sub[$a] cmp $sub[$b] || $a <=> $b } (0 .. $#msgs)) {
 print $msgs[$i];
}
```

Or more succinctly,

```
#!/usr/bin/perl -n00
# bysub2 - awkish sort-by-subject
BEGIN { $msgno = -1 }
$sub[++$msgno] = (/^Subject:\s*(?:Re:\s*)*(.*)/mi)[0] if /^From/m;
$msg[$msgno] .= $_;
END { print @msg[ sort { $sub[$a] cmp $sub[$b] || $a <=> $b } (0 .. $#msg)
] }
```

How do I find out my hostname, domainname, or IP address?

(contributed by brian d foy)

The `Net::Domain` module, which is part of the standard distribution starting in perl5.7.3, can get you the fully qualified domain name (FQDN), the host name, or the domain name.

```
use Net::Domain qw(hostname hostfqdn hostdomain);

my $host = hostfqdn();
```

The `Sys::Hostname` module, included in the standard distribution since perl5.6, can also get the hostname.

```
use Sys::Hostname;

$host = hostname();
```

To get the IP address, you can use the `gethostbyname` built-in function to turn the name into a number. To turn that number into the dotted octet form (a.b.c.d) that most people expect, use the `inet_ntoa` function from the `Socket` module, which also comes with perl.

```
use Socket;

my $address = inet_ntoa(
 scalar gethostbyname( $host || 'localhost' )
 );
```

How do I fetch a news article or the active newsgroups?

Use the `Net::NNTP` or `News::NNTPClient` modules, both available from CPAN. This can make tasks like fetching the newsgroup list as simple as

```
perl -MNews::NNTPClient
  -e 'print News::NNTPClient->new->list("newsgroups")'
```

How do I fetch/put an FTP file?

(contributed by brian d foy)

The LWP family of modules (available on CPAN as the libwww-perl distribution) can work with FTP just like it can with many other protocols. LWP::Simple makes it quite easy to fetch a file:

```
use LWP::Simple;

my $data = get( 'ftp://some.ftp.site/some/file.txt' );
```

If you want more direct or low-level control of the FTP process, you can use the Net::FTP module (in the Standard Library since Perl 5.8). It's documentation has examples showing you just how to do that.

How can I do RPC in Perl?

(contributed by brian d foy)

Use one of the RPC modules you can find on CPAN (http://search.cpan.org/search?query=RPC&mode=all).

AUTHOR AND COPYRIGHT