timeserver.pl

simple Perl TCP/IP server access by telnet localhost 4242

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 4242, Listen => SOMAXCONN) or die;

while ($c = $server->accept()) {
    printf STDERR "[Connection from %s]\n", $c->peerhost;
    print $c scalar localtime,"\n";
    close $c;
}
```

timeclient.pl

simple Perl TCP/IP client

```
use IO::Socket;
$server_host =  $ARGV[0] || 'localhost';
$server_port = 4242;
$c = IO::Socket::INET->new(PeerAddr => $server_host, PeerPort  => $server_port) or die;
$time = <$c>;
close $c;
print "Time is $time\n";
```

webget.pl

fetch files via http from the webserver at the specified URL see HTTP::Request::Common for a more general solution

```
use IO::Socket;
foreach $url (@ARGV) {
    $url =~ /http:\/\/([^\/]+)(:(\d+))?(.*)/ or die;
    $c = IO::Socket::INET->new(PeerAddr => $1, PeerPort => $2 || 80) or die;
    # send request for web page to server
    print $c "GET $4 HTTP/1.0\n\n";
    # read what the server returns
    my @webpage = <$c>;
    close $c;
    print "GET $url =>\n", @webpage, "\n";
}
```

webserver-404.pl

list to port 2041 for incoming connections print then details to stdout then send back a 404 status code

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 2041, ReuseAddr => 1, Listen => SOMAXCONN) or die;

print "Access this server at http://localhost:2041/\n\n";

while ($c = $server->accept()) {
    printf "HTTP request from %s =>\n\n", $c->peerhost;
    while ($request_line = <$c>) {
        print "$request_line";
        last if $request_line !~ /\S/;
    }
    print $c "HTTP/1.0 404 This webserver always returns a 404 status code\n";
    close $c;
}
```

webserver-200.pl

list to port 2041 for incoming connections print then details to stdout then send back a 404 status code

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 2041, ReuseAddr => 1, Listen => SOMAXCONN) or die;

print "Access this web server at http://localhost:2041/\n\n";

$content = "Everything is OK - you will pass COMP[29]041.\n";

while ($c = $server->accept()) {
    printf "HTTP request from %s =>\n\n", $c->peerhost;
    while ($request_line = <$c>) {
        print "$request_line";
        last if $request_line !~ /\S/;
    }

    # print header
    print $c "HTTP/1.0 200 OK\n";
    print $c "Content-Type: text/plain\n";
    printf $c "Content-Length: %d\n\n", length($content);

    print $c $content;
    close $c;
}
```

webserver-too-simple.pl

return files in response to incoming http requests to port 2041 note does not check the request is well-formed or that the file exists also very insecure as pathname may contain ..

```
use IO::Socket;

print "Access this server at http://localhost:2041/\n\n";

while (1) {
    $server = IO::Socket::INET->new(LocalPort => 2041, ReuseAddr => 1, Listen => SOMAXCONN) or die;
    while ($c = $server->accept()) {
        my $request = <$c>;
        print "Connection from ", $c->peerhost, ": $request";
        $request =~ /^GET (.+) HTTP\/1.[01]\s*$/;
        print "Sending back /home/cs2041/public_html/$1\n";
        open my $f, '<',"/home/cs2041/public_html/$1";
        $content = join "", <$f>;
        close $f;
        print $c "HTTP/1.0 200 OK\n";
        print $c "Content-Type: text/html\n";
        printf $c "Content-Length: %d\n\n", length($content);
        print $c $content;
        close $c;
    }
}
```

webserver-mime-types.pl

return files in response to incoming http requests to port 2041, determine appropriate mime type using /etc/mime.types

```
use IO::Socket;

print "Access this server at http://localhost:2041/\n\n";

open my $mt, '<', "/etc/mime.types" or die "Can not open /etc/mime.types: $!\n";
while ($line = <$mt>) {
    $line =~ s/#.*//;
    my ($mime_type, @extensions) = split /\s+/, $line;
    foreach $extension (@extensions) {
        $mime_type{$extension} = $mime_type;
    }
}
close $mt;
while (1) {
    $server = IO::Socket::INET->new(LocalPort => 2041, ReuseAddr => 1, Listen => SOMAXCONN) or die;
    while ($c = $server->accept()) {
        print "waiting for connection";
        my $request = <$c>;
        last if !$request;
        printf "Connection from %s, request: $request", $c->peerhost;
        my $content_type = "text/plain";
        my $status_line = "400 BAD REQUEST";
        my $content = "";

        if (my ($url) = $request =~ /^GET (.+) HTTP\/1.[01]\s*$/) {
            # remove any occurences of .. from pathname to prevent access outside 2041 directory
            $url =~ s/(^|\/)\.\.(\/|$)//g;
            my $file = "/home/cs2041/public_html/$url";
            $file .= "/index.html" if -d $file;

            print "$file requested\n";
            if (open my $f, '<', $file) {
                my ($extension) = $file =~ /\.(\w+)$/;
                $status_line = "200 OK";
                $content_type = $mime_type{$extension} if $extension && $mime_type{$extension};
                $content = join "", <$f>;
            } else {
                $status_line = "404 FILE NOT FOUND";
                $content = "File $file not found\n";
            }
        }
```

webget-slow.pl

```
use IO::Socket;
foreach (@ARGV) {
    $url =~ /http:\/\/([^\/]+)(:(\d+))?(.*)/ or die;
    $c = IO::Socket::INET->new(PeerAddr => $1, PeerPort => $2 || 80) or die;
    # send request for web page to server
    sleep 3600;
    print $c "GET $4 HTTP/1.0\n\n";
    # read what the server returns
    my @webpage = <$c>;
    close $c;
    print "GET $url =>\n", @webpage, "\n";
}
```

webserver-parallel.pl

return files in response to incoming http requests to port 2041 access by http://localhost:2041/ this version handles incoming request in a child process

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 2041, ReuseAddr => 1, Listen => SOMAXCONN) or die;

print "Access this server at http://localhost:2041/\n\n";

while ($c = $server->accept()) {
    if (fork() != 0) {
        # parent process goes to waiting for next request
        close($c);
        next;
    }
    # child processes request
    my $request = <$c>;
    printf "Connection from %s, request: $request", $c->peerhost;
    if (my ($url) = $request =~ /^GET (.+) HTTP\/1.[01]\s*$/) {
        # remove any occurences of .. from pathname to prevent access outside 2041 directory
        $url =~ s/(^|\/)\.\.(\/|$)//g;
        my $file = "/home/cs2041/public_html/$url";
        $file .= "/index.html" if -d $file;
        if (open my $f, '<', $file) {
            print $c "HTTP/1.0 200 OK\nContent-Type: text/html\n\n", <$f>;
        } else {
            print $c "HTTP/1.0 404 FILE NOT FOUND\nContent-Type: text/plain\n\nFile $file not found\n";
        }
    } else {
        print $c "HTTP/1.0 400 BAD REQUEST\nContent-Type: text/plain\n\nBAD REQUEST\n";
    }
    close $c;
    # child must terminate here otherwise it would compete with parent for requests
    exit 0;
}
```

webserver-simple-cgi.pl

return files in response to incoming http requests files with suffix .cgi executed and output returned only GET method supported assumes application/x-www-form-urlencoded data so CGI.pm won't work

See http://search.cpan.org/dist/HTTP-Server-Simple/ for a much more general solution

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 2041, ReuseAddr => 1, Listen => SOMAXCONN) or die;

print "Access this server at http://localhost:2041/\n\n";

while ($c = $server->accept()) {
    my $request = <$c>;
    if ($request =~ /^GET (.+) HTTP\/1.[01]\s*$/) {
        my $url = $1;
        $url =~ s/(^|\/)\.\.(\/|$)//g;
        if ($url =~ /^(.*\.cgi)(\?(.*))?$/) {
            my $cgi_script = "/home/cs2041/public_html/$1";
            $ENV{SCRIPT_URI} = $1;
            $ENV{QUERY_STRING} = $3 || '';
            $ENV{REQUEST_METHOD} = "GET";
            $ENV{REQUEST_URI} = $url;
            print $c "HTTP/1.0 200 OK\n";
            print $c `$cgi_script` if -x $cgi_script;
        } else {
            my $file = "/home/cs2041/public_html/$url";
            $file .= "/index.html" if -d $file;
            if (!-e $file) {
                print $c "HTTP/1.0 404 FILE NOT FOUND\nContent-Type: text/plain\n\nFile $file not found\n";
            } else {
                print $c "HTTP/1.0 200 OK\nContent-Type: text/html\n\n";
                open my $f, '<', $file or next;
                print $c (<$f>);
                close $f;
            }
        }
    } else {
        print $c "HTTP/1.0 400 BAD REQUEST\nContent-Type: text/plain\n\nBAD REQUEST\n";
    }
    close $c;
}
```

webserver-cgi.pl

return files in response to incoming http requests files with suffix .cgi executed and output returned
GET & POST requests handled assumes application/x-www-form-urlencoded data so CGI.pm won't work

See http://search.cpan.org/dist/HTTP-Server-Simple/ for a much more general solution

```
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => 2041, ReuseAddr => 1, Listen => SOMAXCONN) or die;

print "Access this server at http://localhost:2041/\n\n";

while ($c = $server->accept()) {
    my $request = <$c>;
    printf "Connection from %s, request: $request", $c->peerhost;
    my $content_length = 0;
    while (<$c>) {
        print;
        $header_field{$1} = $2 if /(\S+):\s*(.*)/;
        last if /^\s*$/;
    }
    if ($request =~ /^(GET|POST) (.+) HTTP\/1.[01]\s*$/) {
        my $method = $1;
        my $url = $2;
        $url =~ s/(^|\/)\.\.(\/|$)//g;
        if ($url =~ /^(.*\.cgi)(\?(.*))?$/) {
            my $cgi_script = "/home/cs2041/public_html/$1";
            my $parameters = '';
            if ($method eq 'GET') {
                $parameters = $3 if $3;
            } else {
                read($c, $parameters, $header_field{'Content-Length'});
            }
            print $c "HTTP/1.0 200 OK\n";
            print "Running: echo '$parameters'|$cgi_script\n";
            # provide a minimal set of environment variables
            %ENV = (CONTENT_LENGTH => length $parameters,
                    CONTENT_TYPE => 'application/x-www-form-urlencoded',
                    REQUEST_METHOD => $method,
                    REQUEST_URI => $url,
                    SCRIPT_NAME => $cgi_script);
            # obvious security hole here from shell meta-characters in parameters
            print $c `echo '$parameters'|$cgi_script`;
        } else {
            my $file = "/home/cs2041/public_html/$url";
            $file .= "/index.html" if -d $file;
            if (!-e $file) {
                print $c "HTTP/1.0 404 FILE NOT FOUND\nContent-Type: text/plain\n\nFile $file not found\n";
            } else {
```

persistent.pl

```
use Storable;
$cache_file = "./.cache";
%h = %{retrieve($cache_file)} if -r $cache_file;
$h{COUNT}++;
print "This script has now been run $h{COUNT} times\n";
store(\%h, $cache_file);
```

webget-cookies.pl

fetch files via http from the webserver at the specified URL with a simple cookie implementation (no expiry) see HTTP::Request::Common for a more general solution

```
use Storable;

$cookies_db = "./.cookies";
%cookies = %{retrieve($cookies_db)} if -r $cookies_db;

use IO::Socket;
use IO::Socket::SSL;

foreach (@ARGV) {
    my ($protocol, $host, $port, $path) = /(https?):\/\/([^\/:]+)(?::(\d+))?(.*)/ or die;
    if ($protocol eq "http") {
        $c = IO::Socket::INET->new(PeerAddr => $host, PeerPort  => $port || 80) or die;
    } else {
        $c = IO::Socket::SSL->new(PeerAddr => $host, PeerPort  => $port || 443) or die;
    }
    print $c "GET $path HTTP/1.0\n";
    foreach $domain (keys %cookies) {
        next if $host !~ /$domain$/;
        foreach $cookie_path (keys %{$cookies{$domain}}) {
            next if $path !~ /^$cookie_path/;
            foreach $name (keys %{$cookies{$domain}{$path}}) {
                print $c "Cookie: $name=$cookies{$domain}{$path}{$name}\n";
                print STDERR "Sent cookie $name=$cookies{$domain}{$path}{$name}\n";
            }
        }
    }
    print $c "\n";
    while (<$c>) {
        last if /^\s*$/;
        next if !/^Set-Cookie:/i;
        my ($name,$value, %v) = /([^=;\s]+)=([^=;\s]+)/g;
        my $domain = $v{'domain'} || $host;
        my $path = $v{'path'} || $path;
        $cookies{$domain}{$path}{$name} = $value;
        print STDERR "Received cookie $domain $path $name=$value\n";
    }
    my @webpage = <$c>;
    print STDOUT @webpage;
}

store(\%cookies, $cookies_db);
```

simple_cookie.cgi

retrieved value stored for x in cookie if there is one increment and set the cookie to this value

```
$x = 0;
if (defined $ENV{HTTP_COOKIE} && $ENV{HTTP_COOKIE} =~ /\bx=(\d+)/) {
    $x = $1 + 1;
}
print "Content-type: text/html
Set-Cookie: x=$x;

<html><head></head><body>
x=$x
</body></html>";
```

simple_cookie.cgipm.cgi

retrieves value stored for x in cookie if there is one increment and set the cookie to this value

```
use CGI qw/:all/;
use CGI::Cookie;

%cookies = fetch CGI::Cookie;
$x = 0;
$x = $cookies{'x'}->value if $cookies{'x'};
$x++;
print header(-cookie=>"x=$x"), start_html('Cookie Example'), "x=$x", end_html;
```