# User Authentication in Perl

## 1. Using .htaccess

Check out: http://www.wpi.edu/Academics/CCC/Help/Unix/Webdev/htaccess.html

The above site is pretty self-explanatory. The main steps if you want to allow two users with username/password as matt/matt and greg/greg to access the documents in a directory say: http://www.wpi.edu/~mmani/perlSamples/authenticationSamples/sample1

Go to that directory, and create a .htaccess file – it could look like:

```
AuthUserFile /home/mmani/.htpasswd
AuthType Basic
AuthName "Nifty Little Title"
require valid-user
```

The above file says – the password file for the users will be kept in a file called /home/mmani/.htpasswd, and it says that authentication is required for all files in this directory. You can configure permissions for different files using a .htaccess like –

```
AuthUserFile /home/mmani/.htpasswd
AuthType Basic
AuthName "Nifty Little Title"

<Files test.pl>
require valid-user
</Files>

<Files test1.html>
require user matt
</Files>
```

Note – suppose there are 3 files in the directory – test.html, test.pl and test1.html and 2 users – matt and greg. Then anyone can access test.html without any authentication; only matt can access test1.html; both matt and greg can access test.pl

Note that HTTP is a stateless protocol, but still the user needs to login only once, and then the user can perform multiple operations with the server. No logins will be required unless the browser window is closed. This works by the browser sending the authentication information every time it sends a request to the server.

The following example perl script shows how to find out the user name while executing the script.

```
#! /usr/local/bin/perl

use CGI::Carp qw(fatalsToBrowser warningsToBrowser);

print "Content Type: text/html \n\n";
```

```
print <<ENDHTML;
<html>
<head><title>Testing AUthentication</title></head>
<body>This is a simple test for authentication using htaccess
ENDHTML

print "<br>user = $ENV{qw(REMOTE_USER)}<br>\n";

print <<ENDHTML;
</body>
</html>
ENDHTML
```

**2. Using Cookies**

We will show a simple usage of setting cookies, and retrieving the value from cookies. Note that when we use cookies, we can validate a user by testing against a set of users stored in the database, and we can set the corresponding user name, and use this for further operations. Let us do the following – create two files in a directory as:

File test.pl –

```
#! /usr/local/bin/perl

use CGI::Carp qw(fatalsToBrowser warningsToBrowser);
use CGI::Cookie;

$cookie1 = new CGI::Cookie(-name=>'user',-value=>'matt');

print "Set-Cookie: $cookie1\n";
print "Content Type: text/html \n\n";

print <<ENDHTML;
<html>
<head><title>Testing Cookie</title></head>
<body>This is a simple test for authentication using cookie<br>
ENDHTML

%cookies = fetch CGI::Cookie;
foreach $key (keys %cookies) {
  $currCookie = $cookies{$key};
  $currValue = $currCookie->value;
  print "<br> value = $currValue<br>\n";
}

print <<ENDHTML;
</body>
</html>
ENDHTML
```

Note how a new cookie is created, and then it is sent in the response header. So the browser now has the cookie, and it sends the cookie to the server with every request till the browser is closed. For instance, consider accessing the file test1.pl from the browser.

```
#! /usr/local/bin/perl

use CGI::Carp qw(fatalsToBrowser warningsToBrowser);
use CGI::Cookie;

print "Content Type: text/html \n\n";

print <<ENDHTML;
<html>
<head><title>Testing Cookie</title></head>
<body>This is a simple test for authentication using cookie<br>
ENDHTML

%cookies = fetch CGI::Cookie;
foreach $key (keys %cookies) {
  $currCookie = $cookies{$key};
  $currValue = $currCookie->value;
  print "<br> value = $currValue<br>\n";
}

print <<ENDHTML;
</body>
</html>
ENDHTML
```