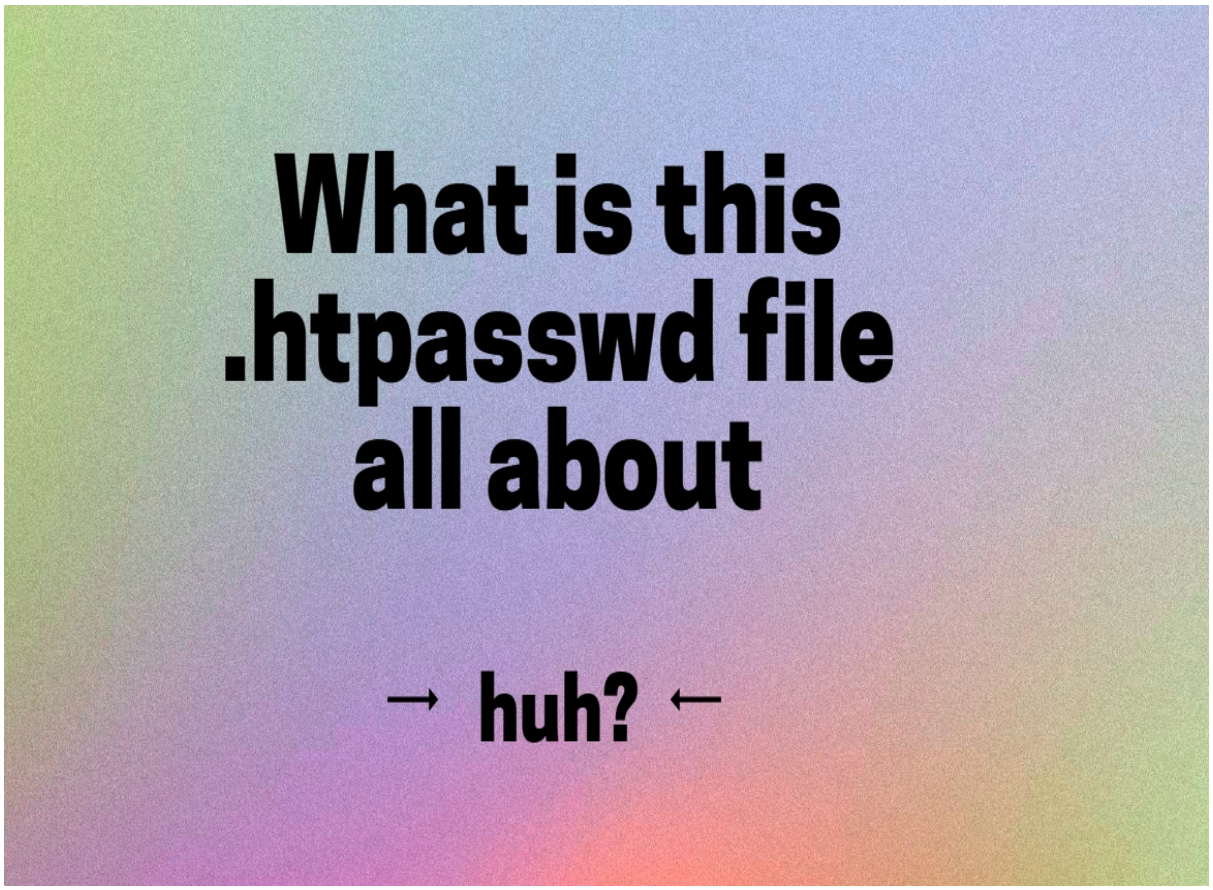


You don't have to use Apache's .htpasswd authentication.

Let me be straight up with you. There are numerous ways to authenticate users to your site. This means that if you are using Apache as your web server, you don't have to use the **.htpasswd** file. This is the only mechanism by which you can password protect pages, files, or folders.

"So, Clyde, why use Apache's password protection techniques, especially if other techniques exist?"

Great question.

The biggest reason to use Apache's authentication techniques is that the logic resides on the server. This means that valid usernames and passwords are almost never shared or accessible by the browser. Nor are they stored in HTML or front-end files, as is the case with other methods.

It's true that developers can code their own authentication schemes. But this can be dangerous because developers are not always security experts, which can lead to insecure applications. The authentication features built into Apache are known to be secure because they have stood the test of time and have been tested ad-nauseam.

Another great advantage of using Apache .htpasswd authentication is that it supports the Basic and Digest authentication protocols straight out of the box.

What is the .htpasswd file?

.htpasswd is just a flat file.

This just means that it is a plain text file. It contains a list of usernames and passwords. The usernames are stored in plain text, but the passwords are hashed.

The file is used by web server software such as Apache or Nginx in order to verify the users through HTTP Basic Authentication.

Nothing more. Nothing less.

Simple, right?

Creating the file

The `.htpasswd` file can be created using `htpasswd` command or the `touch` command or a text editor. But the easiest way is using the `htpasswd` command.

Adding usernames and passwords

You can add usernames and passwords directly or from the command prompt (we have seen this in previous lectures).

However, did you know that you can also update the file dynamically? For example, with PHP, you can hash the password and add it to the **.htpasswd** file, as follows:

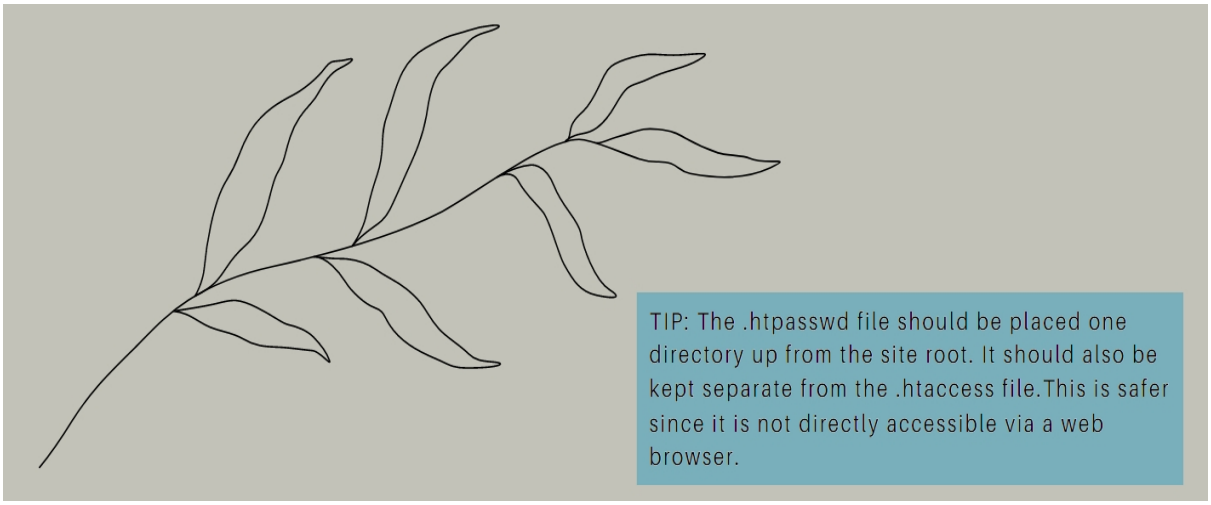
```
1. <?php

1.
2.  $username= 'wally';
3.  $pass = 'secret';
4.  $hash = base64_encode(sha1($pass, true));
5.  $credentials= $username.$hash;
6.  file_put_contents('.htpasswd', $credentials);
7.
8. ?>
```

When should I use the .htpasswd file?

Basically, whenever you want a page or resource to be kept private. Some examples I can think of are as follows:

- Secure an admin page.

- During the development, testing, and creation of your website. This makes sense, right? When you are editing pages on your site, you would not want users to be able to access the work in progress.
- Secure any private sections on your site that you don't want to be made available to general users. For example, you may only want some pages and content to be available to paid members.
- You may only want one client (or a handful of people) to see your site.



Are there any downsides to using Apache's .htpasswd authentication?

Of course. There is never a perfect solution. Perfection does not exist.

Using Apache's **.htpasswd** authentication does not provide any form of brute-force protection. This means that an attacker can make lots of simultaneous attempts at sending a password and Apache will respond with success/failure as soon as it possibly can.

Another problem with using .**htpasswd** authentication is that there's a practical limit to how many users you can put in one password file. Remember, the username and password must be verified every time you request a document from the server, which means that the Apache server has to open up the **.htpasswd** file and go down the list of users until it finds a match. The Apache server has to do this every time a page is loaded.

Bottom line: Once you start getting a lot of users, this can slow things down. (Advanced: if you get to this point, you can then change the authentication process.

For example, you don't have to use a **.htpasswd** file. You could use a database or DBM files, which will greatly improve the retrieval and matching process for a large user base.)

Conclusion

Don't get lost in all of these details.

If you can take these **5 points** away from all of this, I have done my job:

- The Internet never used to have a concept of authentication. All data was public.
- When 1996 came along, HTTP created the concept of authentication.
- The original AJAX object provided Basic Authentication very easily. `xhr.open(method, url, async, [USERNAME, PASSWORD]);`
- When you use an Apache server (for example, you have a PHP backend), then you can use the Apache web server to manage the Basic Authentication process for you. In other words, you don't have to write your own PHP application logic to perform authentication.
- One such option in Apache is to use the **.htpasswd authentication** technique, which uses HTTP Basic Authentication. You can send the username and password credentials using AJAX to the Apache server, and Apache will handle the rest.

That's it.

See you in the next lecture.