# About Apache mod_userdir

Joe Doupnik

20 March 2018

A short while ago on the lists there were queries and comments about venerable Apache module mod_edir which would permit using ~username in URLs to look into a user's eDir home directory. Alas, module mod_edir was for NetWare and times have changed.

There is a regular Apache module, mod_userdir, which works roughly similarly. It too employs ~username in a URL to refer to a user's home directory (which we will see is really just any directory of name username). The module appears in the Apache documentation, the same text for both Apache v2.2 and current v2.4. Here is a snippet from the manual page, just a portion to get us going:

The `UserDir` directive sets the real directory in a user's home directory to use when a request for a document for a user is received. *Directory-filename* is one of the following:

- The name of a directory or a pattern such as those shown below.
- The keyword `disabled`. This turns off *all* username-to-directory translations except those explicitly named with the `enabled` keyword (see below).
- The keyword `disabled` followed by a space-delimited list of usernames. Usernames that appear in such a list will *never* have directory translation performed, even if they appear in an `enabled` clause.
- The keyword `enabled` followed by a space-delimited list of usernames. These usernames will have directory translation performed even if a global disable is in effect, but not if they also appear in a `disabled` clause.

If neither the `enabled` nor the `disabled` keywords appear in the `Userdir` directive, the argument is treated as a filename pattern, and is used to turn the name into a directory specification. A request for `http://www.example.com/~bob/one/two.html` will be translated to:

| UserDir directive used | Translated path |
|---|---|
| UserDir public_html | ~bob/public_html/one/two.html |
| UserDir /usr/web | /usr/web/bob/one/two.html |
| UserDir /home/*/www | /home/bob/www/one/two.html |

The following directives will send redirects to the client:

| UserDir directive used | Translated path |
|---|---|
| UserDir http://www.example.com/users | http://www.example.com/users/bob/one/two.html |
| UserDir http://www.example.com/*/usr | http://www.example.com/bob/usr/one/two.html |
| UserDir http://www.example.com/~*/ | http://www.example.com/~bob/one/two.html |

The easy part is dealing with allow/deny users. By default everyone is allowed. Declaration "UserDir disabled" changes the default to be denied. "UserDir enabled <list of users>" or "UserDir disabled <list of users>" targets users for their particular access.

It turns out the difficult part of this is understanding what the above "Translated path" expressions actually do. Thus let me explain a little (after many experiments and tinkering

with the module source code to report on its actions, plus head scratching and much coffee).

The goal is modify the incoming URI (things after http://example.com/) and convert that URI into a proper file path within the file system (POSIX and NSS). The "UserDir path" declaration can list items which are prefixes (start with /), or suffixes (do not start with /) or have a * in them which are part of the ~username replacement. A * means substitute the literal username rather than the HomeDirectory of a user. The prefix/suffix parts means replace ~username with /prefix/username/suffix. Seems simple, but only at first. Below is my terse algorithmic description, omitting some obscure nuances.

---

If declaration UserDir has a list of path components, try each in turn until an existing filename is found.
The algorithm is of this form --

If the UserDir list is absent then quietly insert prefix /home and suffix public_html. Handy suffix of ./ means "here" and avoids being a subdirectory.

Top of the loop is here.
Get the next item from the UserDir list.
If the list is exhausted or empty then return as-is.

If the item is a prefix then test filename prefix/username for existence.
If that file exists then report it as the filename and quit.
Else go to the loop top.

If the item is a suffix or there is no next item then obtain the username's HomeDir value
and test filename HDvalue/username/suffix (or if no suffix then test file HDvalue/username).
If the HDvalue is absent (no HomeDirectory) then test just filename suffix (or if it is missing then fail and quit).
If that file exists then report it as the filename to employ and quit.
Else go to the loop top.

---

The above description took a lot of decoding/testing/debugging to reveal the inner workings; the source code is far from being clear. There are more nuances but the above covers nearly everything. It's about right (+/- engineering tolerances).

Now, about that HomeDirectory part. With OES there are three H-Ds involved. First there can be a user defined in POSIX space, in /etc/passwd, which specifies a value. In eDirectory there can be two spots: ndsHomeDirectory for NCP work and HomeDirectory for LUM enabled users (whose value is in POSIX file system space). mod_userdir looks to /etc/nsswitch to probe both compat (meaning /etc/passwd) and nam (the NAM cache daemon) for the HomeDirectory value of a username, both spots use POSIX data. We see that only the POSIX aspect of things is used.

Useful hint in passing. If you change a H-D value on the fly then NAM requires we give command "namconfig cache_refresh" and NSCD requires we give command "nscd -i passwd" to clear stale cache values. Best to do both.
Also, when LUM enabling users NAM defaults to using LUM HD value /home/username, but we can modify file /var/lib/novell-lum/namutils.inp to instead use another prefix and a

few other items.

Testing reveals that mod_userdir works for eDirectory defined users which are or are not LUM enabled, for pure Linux users, and for usernames which don't exist as users but are instead just directory names. The difference is some usernames have a readable HomeDirectory value and others do not. This difference is reflected in the algorithm outline written above. For usernames lacking an H-D value we can instead state a prefix as a replacement, or a list of prefixes to try, or a nifty safe&sure mashup pattern of prefix/*/suffix of our choosing. A prefix replaces the user's H-D value. The ordering of items in a UserDir declaration is important because the first file existence match settles matters right then.

To make this work we should remember two things. First, Apache needs permission to look into parts of the file system, which is most often done by <Directory> clauses, but also for NSS volumes user wwwrun needs access rights (iManager, Rights, choose user wwwrun). Second, UserDir declarations cannot reside in <Directory> or <Location> clauses, and thus UserDir applies to Apache as a whole. This means we place UserDir declarations in other files.

SUSE has a pre-packaged file, /etc/apache2/mod_userdir.conf, which is a good spot to place our additions and modifications. Alas, that file also has a <Directory /home/*/public_html> clause, and you may wish to change that clause. We can also simply create a new file in say /etc/apache2/conf.d. The module itself is already loaded by its presence in file /etc/sysconfig/apache2, line APACHE_MODULES="this and that". You may want to remove it from there until you really need it because by default it is active and permits http://example.com/~goodies/whatnot to work for the bad guys. Individual user places can employ .htaccess files to control matters (see the Apache documentation).

That is my short description of the mod_userdir business. I had contemplated writing a new fresh mod_edir module, but mod_userdir does basically what we want and is part of Apache. Testing is the best way of verifying that a configuration does what it ought, and when testing also include a simple web page within subdirectory public_html to catch such references.

Examples follow on the next page.

Examples.

/home has POSIX user named user-linux

/home/NSSVOL has eDir users named user-lum and user-nolum, and a directory named nouser.

Let us try UserDir /aaa /bbb ccc ddd. That's two prefixes and two suffixes. Reading the trace logs, the module first tries to find file /aaa/username, and of course that fails. It moves to the next item in the UserDir list and tries /bbb/username, which also fails.

Moving to the suffixes the module sees no prefix so it tries to obtain one as the real home directory, which works thus far. However, internally it then immediately attaches suffix ccc, which fails, and then suffix ddd, which also fails. So everything fails.

We recall that the UserDir list of paths to consider is read one item at a time, left to right.

If a prefix is not on offer then the user's home directory is requested, followed by any suffixes.

Next, remove the UserDir directive. With username user-linux the module finds

/home/user-linux/public_html. There is that automatic insertion of suffix public_html.

With username user-lum the module finds /home/NSSVOL/user-lum/public_html.

Usernames user-nolum (thus no POSIX home directory) and username nouser (ditto) both return failure.

For fun set the UserDir list to contain only /home/*/./ The ./ part says "here" so don't dig into a subdirectory, and * means the username itself. Only user-linux succeeded and yielded /home/user-linux/ (no public_html).

Lastly set UserDir to hold only /home/NSSVOL/*/./ In this test user-lum, user-nolum, nouser all show their proper home directories (/home/NSSVOL/username, no public_html).