NAME

".urlmon.mfs" − filters for urlmon used for www−page content monitoring

DESCRIPTION

`.urlmon.mfs` includes the 'print' and 'expand' filters. You can instruct **urlmon** to load the filter's code from current working directory by putting at the beginning of your urlmon's database file following line:

```
CODE=$ENV{'PWD'}/.urlmon.mfs
```

If the above doesn't work, then:

```
CODE=/<absolute>/<path>/.urlmon.mfs
```

For more details see the documentation of **urlmon**.

The '**print**' filter just prints out the fetched HTML page. It doesn't require arguments.

The remaining documentation deals with the '**expand**' filter which is a general purpose content filter aimed to produce reports and a summary of reports from monitoring. It examines differences (see the paragraph NOTES below) between the previous and present text content of the monitored URL and reports them! Alternatively, it can pre-filter a HTML document and make a list of included URLs (hyperlinks) which will be retrieved, their text content filtered, and eventually reported. This is the most powerful feature of that filter! The monitored documents are fetched by API provided by the LWP modules and can be converted to ASCII by 'lynx' or 'links'. The filter can be used in forking mode: `'−F'` option of **urlmon**. Simultaneous reading and updating of files by the child processes is based on rudimentary file locking. This mechanism is not very elegant or efficient, but it works :−)

There are three possibilities for monitoring. Which one is used depends on the first and second arguments passed to the filter (see also the ARGUMENTS description):

A) You want to monitor the text content of a given page and you don't want to apply filtering

In this case the first and second arguments can be skipped, alternatively you can say FARGS=−,−

B) You have the situation as in A), but you want to do text content filtering

You must put as second argument the same URL as given in the 'URL=' variable of the urlmon database file:

URL=http://www.example.org/pub/test.html MOD=abcdefghijklmnopqrstuvwxyz FILTER=expand FARGS=<prefilterstr>&<postfilterstr>,http://www.example.org/pub/test.html

C) You want to monitor a list of specific hyperlinks as extracted from the target URL, whereas new members in the list should be retrieved and filtered

First you need to apply a pre-filter, which is defined by regular expressions, to capture the wanted hyperlinks found in the observed page. Consider the targeted URL used in case B) as follows:

URL=http://www.example.org/pub/test.html MOD=abcdefghijklmnopqrstuvwxyz FILTER=expand FARGS=details.cgi&<postfilterstr>,−

Assume that you expect to find in the file ''test.html'' lines with links which point to some script called ''details.cgi'':

```
<HTML><BODY>
    This is <A href="details.cgi?quest=1"> First  </A>
    This is <A href="details.cgi?quest=2"> Second </A>
    Hi everybody!
</BODY></HTML>
```

Now, if you are interested in these links (and aim to request and evaluate them), you can make a list of the wanted URL's by using the pre-filter ''details.cgi'' as defined in FARGS above. After processing the HTML code in ''test.html'', the '**expand**' filter will find and request following list:

```
http://www.example.org/pub/details.cgi?quest=1
http://www.example.org/pub/details.cgi?quest=2
```

Note that the string ''http://www.example.org/pub'' is automatically put in front of every listed item. Or to be precise, an extracted hyperlink is appended to the monitored URL-string stripped from its basename.

In difference, if the lines with links in the above example look like

```
This is <A href="cgi/details.cgi?quest=1"> First  </A>
This is <A href="cgi/details.cgi?quest=2"> Second </A>
```

you need to give a second argument in FARGS defining the in-front part of the extracted links:
URL=http://www.example.org/pub/test.html MOD=abcdefghijklmnopqrstuvwxyz FILTER=expand FARGS=details.cgi&<postfilterstr>,http://www.example.org/

The resulting list will be:

```
http://www.example.org/cgi/details.cgi?quest=1
http://www.example.org/cgi/details.cgi?quest=2
```

These documents are retrieved then, and filtered by <postfilterstr>.

In all examples in C), the string ‘‘details.cgi’’ plays the role of a pre-filter submitted as FARGS argument. You are not restricted to use only one pre-filter, but as many as you want. They apply to the HTML content of the monitored URL, and are used to find out the wanted links in the HTML document. Hence, these pre-filters behave a little bit differently compared to the cases A) and B)! Furthermore, they are applied two times at different steps in the procedure of determining the list of documents to be retrieved. Be careful when using more than one pre-filter − define the filters such that they include only regular expressions which are found at the right-hand side of the http-link looking for! Still, if you want that the pre-filters are applied only one time, and that the filter works as a regex to the whole input line, then the token ‘‘_LO:’’ must be put in front of the pre-filter string (alias ‘‘link only’’, or LO). Using again the last example, a link only filter definition could be:
URL=http://www.example.org/pub/test.html MOD=abcdefghijklmnopqrstuvwxyz FILTER=expand FARGS=_LO:/This.is/&<postfilterstr>,http://www.example.org/

One link per line is expected when using a LO-filter.

Note, that in case C) the comparison between the last and present content of the monitored page is actually a comparison between the last and present list of extracted URL’s. Also, in the discussed examples we implicitly assumed that the last list is missing as we just start to monitor the page.

```
Enjoy!
```

ARGUMENTS

Arguments which are skipped (undetermined) receive default values! If you want to skip arguments between determined arguments, then you have to put the ’−’ character, as for example: FARGS=<arg1>,−,−,<arg4>

For some arguments the empty string ’’ or "", which means an empty argument value, can be given! In such cases ’−’ and ’’ are not the same!

The arguments used by the ’**expand**’ filter, and ordered by their index, are described as follows:

1) A string defining pre− and post-filters. Although from the point of view of **urlmon** the whole code of ’**expand**’ is considered as an external filter, the actual content filtering is determined by this argument. Hence, the argument is called ‘‘filter(s)’’ in this documentation. All filters are applied by using the *grep()* function. They are separated by unescaped ’;’ character not followed by ’;’. Thus ’;;’ is not interpreted as a filter separator, and one can use it in braced blocks (i.e. { BLOCK }, see below) instead of ’;’. A combination of pre− and post-filters is a complex string consisting of two substrings separated by unescaped ’&’ not followed by ’&’ which means that you can have ’&&’ in braced block or elsewhere.

The pre-filters can be applied to the content before the old and new contents are compared (case A and B in the examples above), or can be used for extracting wanted links (case C). In latter case, the URL’s content is processed line by line. Only when the pre-filter is passed, the line is further evaluated. It is split into strings separated by the substring ’ href=’, and then the pre-filter is applied again to every single string. This behavior can be changed by defining a pre-filter as ‘‘link only’’. In order to do so put in front of your pre-filter the token ‘‘_LO:’’ (look at the example in C). Consequently, the prefilter is applied to whole lines without splitting them.

The post-filters are used to filter the text content of a retrieved page. For post-filtering the content is joined into a single record (i.e. single string).

Ultimately, the pre− and post− filters have generally a slightly different syntax since the first are applied by *grep()* in list context, the latter in scalar context, respectively. This complication has the advantage that by using post-filters one can easily discard a whole record based on various criteria, as for example looking for given keywords in the record. For both, pre− and post-filters, every single filter is automatically wrapped within '//' (e.g. <string> is wrapped to /<string>/) if it doesn't include an unescaped '/' and if it isn't defined in block. One can also give already wrapped filter(s), e.g. /<string1>/;!/<string2>/i; etc., or a mixture of wrapped and unwrapped filters. Any complex filter(s) must be defined as a single string without space characters!! Every filter from the complex filter string is implemented as block expression, i.e filtering is applied by using: '@list = grep { /<filterstr>/ } `@list`;' Eventually, you could apply also filters which are explicitly defined in a block! This allows writing sophisticated filter code. Also escaped commas can be used. Before applying the filters commas are then unescaped which means that for an escaped comma one needs to have '\\,'.

```
                     EXAMPLES FOR VALID PRE−FILTERS:

 (?i)BoY −> will be wrapped to /(?i)BoY/, equal to /BoY/i
 !/girl/i −> delete lines containing the word 'girl'
 (?i)(Woman|girl) −> keep lines containing the word 'woman' or 'girl'
 s/sex/SEX/g||/.*/;!/violence/i −> substitute global + abandon violence
            ˆˆˆˆˆˆ−−−> this part is needed in case that the whole list
                       corresponding to the record does not contain the
                       word 'sex' but you want to keep the record anyway
 {$I0++<=9&&$_>100} −> the first 10 lines of the record; if these are
                       numbers, then include only numbers > 100 ; $I0 is
                       declared as a global variable! Be careful whether
                       it is used elsewhere!

 Using /./ instead of /.*/ in the regular expressions will discard all
 blank lines!

                     EXAMPLES FOR VALID POST−FILTERS:

 !/bad/ −> delete the whole record if it contains the word 'bad'
 /(Happy.*Days|Days.*Happy)/si −> keep the record if it contains
 somewhere in it the two words 'Happy' and 'Days', case insensitive
 s/EVIL/GOOD/mg||/.*/ −> global change of 'EVIL' into 'GOOD'
 s/ˆ.*ADVERTisement.*\n//mgi||/.*/ −> delete lines with 'advertisement'
 s/ˆ.*ADVERTisement.*\n//mgi −> this is wrong, because if there is not
 such word in the record, it will be discarded by grep!!
 s/ˆ(?:(?!Love).)*\n//mg||/.*/;s/ˆ(?:(?!Peace).)*\n//mg||/.*/ −> keep
 lines which contain both 'Love' and 'Peace'
 /Luck(?=(?:(?!Trouble).)*$)/si −> Luck which is not followed by Trouble
 /smart/si&/ˆ(?:(?!soft).)*$/si −> smart but not soft

                     COMBINATIONS OF PRE− AND POST−FILTERS:

 simple −> this applies a pre−filter only
 &easy −> this applies a post−filter only
 simple&easy −> it's the same as /simple/&/easy/
 (?i)simple;!/complex/i&easy;s/ˆ(?:(?!dummy).)*\n//mig||/.*/
 (?i)simple;!/complex/i;&easy;s/ˆ(?:(?!dummy).)*\n//mig||/.*/; −> the same
 as previous but with explicit separator ';' put after every one filter
```

Well, this is not all about filters. One can put them (or part of them) in a file which is parsed and included into the complex filter string. And that's really fun! To include such file(s) the token '_FiFi_:' should be given somewhere in the complex filter string e.g.:

```
        simple&easy;_FiFi_:~/urlmon/myfilters;
```

Then all of the filters defined in '~/urlmon/myfilters' will be pasted exactly at this place in the filter string. Or another example:

```
        simple;_FiFi_:~/urlmon/addpre;&easy;_FiFi_:~/urlmon/addpost;
```

Finally, one can define all of his filters (pre− and post-filters) in an external file and just say:

```
        _FiFi_:~/urlmon/allmyfilters;
```

It's important to put ';' after the file name! Before pasting the filter file it is parsed such that all blank characters, empty lines and also comments (lines starting with any number of blank characters followed by '#') are removed. Consequently, you can define filters in an external file and use blank characters in the code as opposed to the default situation where the complex filter string should be a single string without blanks. The other rules for writing filters, as previously explained, are still valid. So, separating pre− and post-filters in a filter file is done as usual by putting '&'. But don't forget, at the end, when the complex filter string is returned to the program, it should contain only one separating '&'.

2) Base URL (use '−' for auto-find-out, '' or "" for none). For database requests, if the extracted document's path/URL for the request has an overlapping part with the base URL-string, then this part of the extracted string which is in front of the overlapping is deleted. For example, if the base URL is: ''http://www.example.org/pub/documents'' and the document's path is: ''../somedir/somesubdir/documents/fetchme?id=123'' then the constructed URL for the database request will be: ''http://www.example.org/pub/documents/fetchme?id=123".

3) Directory (for working) where the data files will be saved − the current working directory is default. Paths should be absolute or relative to the working directory. One can also use Perl variables `$ENV`{'HOME'}, `$ENV`{'PWD'}, and also ~/, ~/<mypath>/ etc.

4) Basename of the file in which the results for this URL will be saved (use '−' for auto naming). Generally, three files are created: one with extension '.old', next with extension '.new', and the last with extension '.rep' which will be a link to the report file. The name of the actual report file is generated automatically and consists of the basename, date and time tag, and increasing counter number.

5) File where to save summary of the results (use '−' for auto, '' or "" for none). ''Summary.rep'' is the default (see also the previous argument). The magic word ''STDOUT" (case insensitive) will cause the summary to be sent to the standard output. In this case mail will not be sent and the output will not have a header. Use absolute paths in the filename if you have different directories for different URL entries and you want to be sure that all reports are written to the same file! In case the 3rd argument is given, the relative paths for this file are then relative to it.

6) E−mail address(es) or file with address(es) to send report to. In case you have a summary (see argument 5) the summary is e−mailed. Otherwise the file with the results for this URL is sent. The validity of the address is not checked. Multiple e−mail addresses can be passed to the program by giving a string containing different addresses separated by '&': "user1@host1&user2@host2&user3" etc. When using file with e−mail addresses, write the absolute path, `$ENV`{'HOME'}/<somepath>/<filename>, or at least ./<filename>! E−mail addresses in this file can be separated by blanks, new lines, or '&'. If there are no changes between the old and new documents, mail will not be sent. The mail queue is checked for unique delivery orders such that a report will be sent only once to the user. Hence one can give repeatedly his e−mail address in different URL entries (of the urlmon database) for the same report.

7) Total number of pages in a collection. This argument is needed if you want to observe many pages (URLs) as a single one collected page. As for example, when you monitor an URL with a list of database items (links), but the number of items in the list is limited by the provider, you have to load in sequence all following pages with items to go through the whole list. The solution is to define explicitly additional URL entries in the urlmon's database file for every following page in the provider's generated list. When monitoring a collection of pages and **urlmon** is executed in forking mode, the present argument is essential.

Nevertheless, it has to be defined in any case when pages are collected. The basename of the file where the results are saved (see argument 4) must be the same for all URLs in the collection!

8) Program to use for retrieving the extracted list of items from the monitored URL − like in case **C)** above. It should print out the downloaded HTML documensts to standard output. Such a program could be ’wget’ for example. In this case the wget-options ’−q −O−’ are automatically added on execution.

9) Program to use for converting the HTML document to ASCII. ’lynx’, ’links’, or user-defined program can be used:
  ’lynx’ or /<path>/lynx − use ’lynx’ (default)
  ’links’ or /<path>/links − use ’links’
  /<path>/<myconverter> − user-defined program that converts HTML to ASCII. The program should expects as (only) command-line argument the name of the file to be converted, and should print out the ASCII text to STDOUT. Eventually, a private converter normally serves as an external content filter that presumably does [quite] complicated filtering and executes additional user-defined tasks.

10) Debug level: 1−5 are possible values

NOTES

The difference between the new and old text content of a document is considered to be the additional lines in the new document compared to the old one. If there are no additional lines but only few lines have changed, or alternatively [almost] the whole text is new, then only the changed parts of the documents are reported in the first case; the whole content is reported to be changed in the latter case, respectively. For details on the method of how the boundary is defined between small and big difference, look at the code :−)

ENVIRONMENT VARIABLES

There are following variables which can be defined in the OS environment to further configure the behavior of the program:

URLMON_CTYPE − could be used to override the LC_CTYPE setting and to define the character encoding of the resulting output text.

URLMON_SLEEP_FORNEXT − defines the time (in floating seconds) which the ’expand’−filter waits before fetching the next item on its list of extracted urls. ‘‘Don’t wait’’ is the default behavior.

COPYRIGHT

Copyright (C) 2001−2008, 2011−2014 Dimitar Ivanov, <dimitar.ivanov@mirendom.net>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

You can find the latest version of the program at http://gnu.mirendom.net

SEE ALSO

Check the documentation in the **urlmon** distribution.