

manweb: A CGI script to display man pages



John W. Shipman

2009-10-10 13:13

Abstract

Describes a Web script to display Unix man pages online.

This publication is available in Web form¹ and also as a PDF document². Please forward any comments to tcc-doc@nmt.edu.

Table of Contents

1. Introduction	1
2. The path names of man pages	1
3. Operation of the script	2
4. Internals	3
4.1. Installation	3
4.2. Prologue	3
4.3. Imports	4
4.4. Manifest constants	4
4.5. Main program	6
4.6. processArguments(): Retrieve and check the CGI arguments	7
4.7. findManFile(): Locate the input file	9
4.8. convert: Convert to HTML and output	10
4.9. errorPage	11
4.10. Epilogue	12
5. Error statistics	12
5.1. Run-time type errors	12

1. Introduction

The purpose of this script is to display a selected Unix man page in HTML. It uses the CGI (Common Gateway Interface) protocol to generate and display the web page.

Some consideration is given to security. Arguments passed to the script may contain only certain characters.

2. The path names of man pages

The standard location for man pages is in subdirectories of `"/usr/share/man"`. These subdirectories' names must start with "man". A cursory survey of man page directories in various installs found direct-

¹ <http://www.nmt.edu/tcc/help/lang/python/examples/manweb/>

² <http://www.nmt.edu/tcc/help/lang/python/examples/manweb/manweb.pdf>

ories named `man0p`, `man1`, `man1p`, `man1x`, `man2`, `man2x`, `man3`, `man3p`, `man3x`, `man4`, `man4x`, `man5`, `man5x`, `man6`, `man6x`, `man7`, `man7x`, `man8`, `man8x`, `man9`, `man9x`, `manl`, and `mann`.

The only directory structure that matters is the one on the web server where this script is installed. At this writing (December 2007), the aging `infohost` server has only `man1` through `man9`, plus `mann`, with no suffixes. However, this script will be more liberal, in anticipation of future installs that may be more like client machines.

The language of `man` pages is *groff*; for details, see the project page at gnu.org³.

The actual files are compressed with *gzip*, so their names end with `.gz`.

Here are some examples of actual path names:

```
/usr/share/man/man1/ls.1.gz
/usr/share/man/man1/Magick+-config.1.gz
/usr/share/man/man0p/math.h.0p.gz
/usr/share/man/man3/tan.3.gz
/usr/share/man/man3/Thread::Queue.3pm.gz
/usr/share/man/man3/trace.3x.gz
/usr/share/man/man3/ui.3ssl.gz
/usr/share/man/man3/vars.3pm.gz
/usr/share/man/man1p/nohup.1p.gz
/usr/share/man/man6x/bouncingcow.6x.gz
```

In order to allow access only to legitimate `man` pages, this script will limit access to pathnames of this general form:

```
/usr/share/man/manS/P.SX.gz
```

S

The section identifier. This must be a single digit from 0 to 9 inclusive, optionally followed by either "p" or "x".

P

Valid page names must start with a letter or underbar (`_`). All additional characters must be letters, digits, underbars, or any of the characters `{-, +, ., :}`.

X

An optional suffix that appears in the file name right before the final `.gz`. This suffix, if any, may contain only letters.

3. Operation of the script

To view a `man` page on the Web, use a URL of this form:

```
http://www.nmt.edu/tcc/cgi/manweb.cgi?p=P&s=S&x=X
```

where *P* is the page name, *S* is the optional section name, and *X* is the optional suffix string, as described in Section 2, "The path names of `man` pages" (p. 1).

Here are some examples. The first column shows the part of the URL after the "?"; the second column shows the resulting path name relative to `/usr/share/man/`. Note that all special characters must be escaped using the convention that a character with hexadecimal code *xx* is represented as `"%xx"`.

³ <http://www.gnu.org/software/groff/>

Arguments	Path
p=ls	man1/ls.1.gz
p=Magick%2B%2B%2Dconfig	man1/Magick+-config.1.gz
p=math%2Eh&s=0p	man0p/math.h.0p.gz
s=3&p=Thread%3A%3AQueue&x=pm	man3/Thread::Queue.3pm.gz

If any of the supplied arguments are invalid, or if the specified `man` page does not exist, the script will display an error page. In the latter case, mail will also be sent to `webmaster@nmt.edu` so that broken links can be repaired.

One potential problem is that the set of `man` pages on the server will be different from the set of pages available on clients. Since this script is primarily for basic `man` pages that exist everywhere, we'll deal with that if and when it becomes a problem. One possible solution is to duplicate the set of client `man` pages on `infohost`, and add an option to select the client or server set.

4. Internals

The actual work of reformatting the `man` file into HTML is done by `/usr/bin/man2html`. The `manweb` script is primarily a wrapper that does stringent error checking.

Here is the actual Python script, in `lightweight literate programming`⁴ form.

4.1. Installation

Executable CGI scripts live in directory `"/u/www/docs/tcc/cgi/"`. To install this script, either soft-link that directory to here, or copy the `manweb.cgi` file there.

Online files related to this project:

- `manweb.cgi`⁵: Source of the CGI script.
- `manweb.xml`⁶: DocBook source for this document.

4.2. Prologue

The script starts with the usual "pound-bang line" to make it self-executing, followed by a comment pointing readers back to this documentation.

```

                                                                    manweb.cgi
#!/usr/bin/env python
#=====
# manweb.cgi: Display a man page in HTML. For documentation, see:
#   http://www.nmt.edu/tcc/help/lang/python/examples/manweb/
#-----
```

⁴ <http://www.nmt.edu/~shipman/soft/litprog/>

⁵ <http://www.nmt.edu/tcc/help/lang/python/examples/manweb/manweb.cgi>

⁶ <http://www.nmt.edu/tcc/help/lang/python/examples/manweb/manweb.xml>

4.3. Imports

We'll need two standard Python modules: `sys` for the standard streams, and `os` for the `popen` function to execute commands in a subshell, and the `environ` dictionary containing the currently defined environmental variables.

```
manweb.cgi
#=====
# Imports
#-----

import sys, os
```

Python's `cgi` module takes care of retrieving the arguments from the URL.

```
manweb.cgi
import cgi
```

The Python regular expression module will handle checking the arguments for correct syntax.

```
manweb.cgi
import re
```

4.4. Manifest constants

Next we'll define constants used by the script.

4.4.1. MAN_BASE

The absolute path to the root of the structure containing the `man` pages.

```
manweb.cgi
#=====
# Manifest constants
#-----

MAN_BASE = "/usr/share/man/man"
```

4.4.2. PAGENAME_RE

Regular expressions are used to check the format of the script's arguments. The `PAGENAME_RE` regular expression is used to check the *pagename*.

Important

The pattern ends with the end-of-line anchor (`$`) to ensure that there are no stray unmatched characters at the end of the string. This is a security consideration. Since the page name will become part of a shell command, we need to be sure that no shell metacharacters such as pipe (`|`) are included—that can be a penetration route for evildoers.

```
manweb.cgi
PAGENAME_RE = re.compile (
    r'[_a-zA-Z]'          # Starts with letter or underbar
```

```
r'[-+.:_a-zA-Z0-9]*' # Zero or more: + - . : _ letter digit
r'$' ) # Match the entire string
```

4.4.3. SECTION_RE

The SECTION_RE regular expression is used to test the section number. This regular expression covers all the currently known cases as well as some unlikely combinations such as manlp or mannx; there is no great downside to letting these through, since there won't be any man pages at that path.

manweb.cgi

```
SECTION_RE = re.compile (
    r'[0-9ln]' # Starts with one digit, letter l, or n
    r'[px]?' # Optional suffix
    r'$' ) # Match the entire string
```

4.4.4. SUFFIX_RE

This is the regular expression used to validate the suffix argument: it may contain only letters.

manweb.cgi

```
SUFFIX_RE = re.compile (
    r'[a-z]+' # One or more letters
    r'$' ) # Match the entire string
```

4.4.5. GATEWAY

This is the name of the environmental variable that tells us what gateway interface standard is executing us; it is unset if we're not being called as a CGI script.

manweb.cgi

```
GATEWAY = "GATEWAY_INTERFACE"
```

4.4.6. CGI_VERSION

This constant is used to check for the correct environment: if we are being called as a CGI script, the GATEWAY environmental variable should have this value.

manweb.cgi

```
CGI_VERSION = "CGI/1.1"
```

4.4.7. PAGE_NAME_ARG

This is the name of the argument, passed in the URL, that defines what page the user wants to see.

manweb.cgi

```
PAGE_NAME_ARG = "p"
```

4.4.8. SECTION_ARG

This is the name of the optional argument, passed in the URL, that defines which section contains the desired man page.

manweb.cgi

```
SECTION_ARG = "s"
```

4.4.9. DEFAULT_SECTION

This constant is the default section number when one is not supplied.

manweb.cgi

```
DEFAULT_SECTION = "1"
```

4.4.10. SUFFIX_ARG

This is the name of the optional suffix argument, passed in via the URL, that is appended before the terminal ".gz" of the man file name. The default value is an empty string.

manweb.cgi

```
SUFFIX_ARG = ""
```

4.5. Main program

Here is the main program. The source code comments in square brackets are intended functions for the Cleanroom software development methodology.⁷

manweb.cgi

```
# - - -   m a i n

def main():
    """Main program: display a man page in HTML.

    [ if (this script is being executed by the CGI protocol) and
      (its arguments are valid) and
      (the arguments described an existing man page ->
        sys.stdout += the output of man2html reformatting
                      that page
      else ->
        sys.stdout += an error message as HTML ]
    """
```

We must check that the protocol is CGI, retrieve the arguments, and check them for validity. See Section 4.6, "processArguments(): Retrieve and check the CGI arguments" (p. 7), which does not return unless everything is valid.

manweb.cgi

```
#-- 1 --
# [ if (this script is being executed by the CGI protocol) and
#   (its arguments are valid) ->
#     pageName := the page name argument
#     section := the section number argument, defaulting to "1"
#   else ->
#     sys.stdout += an error message as HTML
#     stop execution ]
pageName, section, suffix = processArguments()
```

Next we build the absolute path name to the input file and check to see if it exists. See Section 4.7, "findManFile(): Locate the input file" (p. 9).

⁷ <http://www.nmt.edu/~shipman/soft/clean/>

```

#-- 3 --
# [ if the man file specified by pagename and section exists ->
#   manPath := the absolute pathname to that file
#   else ->
#     sys.stdout += an error message as HTML
#     stop execution ]
manPath = findManFile ( pageName, section, suffix )

```

At this point we can fire up *man2html*, feed it the file, and route its output back to our standard output stream. Things can still go wrong, however. See Section 4.8, “convert: Convert to HTML and output” (p. 10).

```

#-- 4 --
# [ if manPath specifies a readable, valid man file ->
#   sys.stdout += the output of man2html operating on
#               that file
#   else ->
#     sys.stdout += an error message as HTML ]
convert ( manPath )

```

4.6. processArguments (): Retrieve and check the CGI arguments

This function checks the protocol, retrieves the arguments, and checks them for validity.

```

# - - -   p r o c e s s A r g u m e n t s

def processArguments():
    """Retrieve and validate the arguments.

    [ if (this script is being executed by the CGI protocol) and
      (its arguments are valid) ->
        return (page name argument, section number argument
                defaulting to "1", suffix defaulting to "")
      else ->
        sys.stdout += an error message as HTML
        stop execution ]
    """

```

First we check to see if the CGI protocol is in effect. If so, the environmental variable `GATEWAY` will have the value `CGI_VERSION`; see Section 4.4.5, “GATEWAY” (p. 5) and Section 4.4.6, “CGI_VERSION” (p. 5).

```

#-- 1 --
# [ if environmental variable GATEWAY is defined has the value
#   CGI_VERSION ->
#     I
#   else ->
#     sys.stdout += an HTML error message
#     stop execution ]
try:
    gateway = os.environ [ GATEWAY ]
    if gateway != CGI_VERSION:

```

```

        errorPage ( "Incorrect CGI protocol version." )
    except KeyError:
        errorPage ( "This script must be executed using the CGI "
                    "protocol." )

```

Next we use Python's `cgi` module to extract the arguments following the “?” in the URL, converting them into a `cgi.FieldStorage` instance. For details of this module, see the online documentation⁸.

manweb.cgi

```

#-- 2 --
# [ form := a cgi.FieldStorage instance representing the
#       arguments from the URL used to invoke this script ]
form = cgi.FieldStorage()

```

The `PAGE_NAME_ARG` argument is required; the `SECTION_ARG` is optional and defaults to `DEFAULT_SECTION`; and the `SUFFIX_ARG` is optional and defaults to an empty string.

The `form` instance acts like a dictionary; if an argument was not supplied, attempting to extract it will raise a `KeyError` exception. See Section 4.4.7, “`PAGE_NAME_ARG`” (p. 5).

manweb.cgi

```

#-- 3 --
# [ if (form[PAGE_NAME_ARG] does not exist) or
#     (it exists but is not valid) ->
#     sys.stdout += an error message as HTML
#     stop execution
#     else ->
#     pageName := the corresponding value
try:
    pageName = form[PAGE_NAME_ARG].value
except KeyError:
    errorPage ( "The 'p=PAGENAME' argument is required." )

```

For security reasons, we need to make sure that the page name does not contain any unusual characters that might cause security holes. See Section 4.4.2, “`PAGENAME_RE`” (p. 4) for the regular expression used to check its syntax.

manweb.cgi

```

#-- 4 --
# [ if pageName matches PAGENAME_RE ->
#     I
#     else ->
#     sys.stdout += an error message as HTML
#     stop execution ]
m = PAGENAME_RE.match ( pageName )
if m is None:
    errorPage ( "The 'p=PAGENAME' argument is not valid." )

```

Retrieval and checking of the section name proceeds similarly, except that we have to supply a default value if the section name wasn't specified. See Section 4.4.8, “`SECTION_ARG`” (p. 5) and Section 4.4.9, “`DEFAULT_SECTION`” (p. 6). For the regular expression used to validate the section name, see Section 4.4.3, “`SECTION_RE`” (p. 5).

manweb.cgi

```

#-- 5 --
# [ if form[SECTION_ARG] does not exist ->

```

⁸ <http://docs.python.org/lib/module-cgi.html>

```

#   sectionName := DEFAULT_SECTION
#   else if it exists and is valid ->
#   sectionName := the corresponding value
#   else ->
#   sys.stdout += an error message as HTML
#   stop execution
try:
    section = form[SECTION_ARG].value
    m = SECTION_RE.match ( section )
    if m is None:
        errorPage ( "The 's=SECTION' argument is not valid." )
except KeyError:
    section = DEFAULT_SECTION

```

Also check for the optional suffix argument, which defaults to the empty string. See Section 4.4.10, “SUFFIX_ARG” (p. 6) and Section 4.4.4, “SUFFIX_RE” (p. 5).

manweb.cgi

```

#-- 7 --
# [ if form[SUFFIX_ARG] does not exist ->
#   suffix := ""
#   else if form[SUFFIX_ARG] exists and is valid ->
#   suffix := the corresponding value
#   else ->
#   sys.stdout += an error message as HTML
#   stop execution
try:
    suffix = form[SUFFIX_ARG].value
    m = SUFFIX_RE.match ( suffix )
    if m is None:
        errorPage ( "The 'x=SUFFIX' argument is not valid." )
except KeyError:
    suffix = ""

```

Finally, return the three variable parts of the file name.

manweb.cgi

```

#-- 8 --
return (pageName, section, suffix)

```

4.7. findManFile(): Locate the input file

This function takes the page name and section, builds a full absolute path name, and checks that the file exists.

manweb.cgi

```

# - - -   f i n d M a n F i l e

def findManFile ( pageName, section, suffix ):
    """Locate the man page and verify its existence.

    [ (pageName is a page name as a string) and
      (section is a section number as a string) and
      (suffix is a suffix string) ->
      if that man page exists in that section ->

```

```

        return the absolute path to the page
    else ->
        sys.stdout += an error message as HTML
        stop execution ]
    "" ""

```

First build the absolute pathname to the file. Then see if the file exists. See Section 4.4.1, “MAN_BASE” (p. 4) for the constant that defines the path to the top directory of the man page structure.

manweb.cgi

```

#-- 1 --
# [ manPath := path to the man file for page name
#   (pageName), section (section), and suffix (suffix) ]
manPath = ( "%s%s/%s.%s%s.gz" %
            (MAN_BASE, section, pageName, section, suffix) )

#-- 2 --
# [ if manPath names an existing file ->
#   I
#   else ->
#     sys.stdout += an error message in HTML
#     stop execution ]
if not os.path.exists ( manPath ):
    errorPage ( "No such man page: %s" % manPath )

#-- 3 --
return manPath

```

4.8. convert: Convert to HTML and output

This function attempts to send the file specified by manPath to *man2html*, capture its output, and send it to our standard output.

manweb.cgi

```

# - - -   c o n v e r t

def convert ( manPath ):
    """"Convert a man page to HTML and display it.

    [ manPath is a string ->
      if manPath specifies a readable, valid man file ->
        sys.stdout += the output of man2html operating on
                      that file
      else ->
        sys.stdout += an error message as HTML ]
    "" ""

```

The first complication is that the man file will be compressed with *gzip*, and *man2html* will not uncompress them. We can use *zcat* to uncompress it. So, the command we execute is a pipe of this form:

```
zcat file | man2html
```

manweb.cgi

```

#-- 1 --
command = "zcat %s|man2html" % manPath

```

The standard Python function `os.popen()` executes a shell command in a subprocess and returns a readable file handle that we can use to retrieve its output. If the command fails, we don't find out until we close the file; the `.close()` method returns a Boolean value, `True` if the operation failed, `False` if it succeeded.

manweb.cgi

```
#-- 2 --
# [ pipe := an open file handle for reading the output of
#       command ]
pipe = os.popen ( command )
```

We have to retrieve the output before we can close the pipe and find out whether the operation failed. If it failed, it won't send us any data, so in that case this step does nothing.

manweb.cgi

```
#-- 3 --
# [ sys.stdout += contents of pipe ]
sys.stdout.write ( pipe.read() )

#-- 4 --
# [ if pipe.close returns a false value ->
#   I
#   else ->
#     sys.stdout += an error message in HTML ]
status = pipe.close()
if status:
    errorPage ( "The man2html conversion step for file '%s' "
               "failed." % manPath )
```

4.9. errorPage

This function creates an HTML page telling the user that the operation failed. The argument is a text string giving details of the failure.

manweb.cgi

```
# - - -   e r r o r P a g e

def errorPage ( text ):
    """Write an HTML error message page.

    [ text is a string ->
      sys.stdout += an HTML error message page containing
      text
      stop execution ]
    """
    #-- 1 --
    # [ sys.stdout += CGI headers for generating HTML ]
    print "Content-type: text/html"
    print
    print "<html>"
    print "<head>"
    print "  <title>manweb error</title>"
    print "</head>"
    print "<body>"
    print "<h1><tt>manweb</tt> script error</h1>"
```

```
print "<p>We were unable to process your request: %s" % text
print "</p>"
print "</body>"
print "</html>"
sys.exit(1)
```

4.10. Epilogue

The last lines of the script invoke the `main()`.

`manweb.cgi`

```
if __name__ == "__main__":
    main()
```

5. Error statistics

Here is a list of all errors discovered since first compilation.

5.1. Run-time type errors

These are errors that would have been caught in a more strongly-typed language.

1. In `processArguments()`, the values from the `FieldStorage` constructor are instances, not simple strings; it is necessary to extract their `.value` attribute to get the actual contents.
2. At one point `convert()` was going to take three arguments, including the original page name and section, for better error reporting. I changed my mind but didn't change the definition of this function.