

# homelist.py: A Python script to generate a list of user homepages



John W. Shipman

2009-10-10 14:26



## Table of Contents

1. Requirements .....	1
2. Operation .....	1
2.1. Finding user homepages .....	2
2.2. LDAP considerations .....	3
2.3. <b>crontab</b> setup .....	3
3. Program source .....	4
3.1. The prologue .....	4
3.2. Imported modules .....	4
3.3. Manifest constants .....	5
3.4. <b>createTable()</b> : Set up the XHTML table .....	6
3.5. <b>scanAccounts()</b> : Find all homepages .....	7
3.6. <b>generateAccounts()</b> : Search LDAP for accounts .....	8
3.7. <b>hasHomepage()</b> : Test for the presence of a homepage .....	9
3.8. <b>addRow()</b> : Add a row to the result table .....	10
3.9. <b>addEpilogue()</b> : Add end comments .....	11
3.10. The main program .....	12

## 1. Requirements

---

This document describes a script for generate a current list of the user homepages at the Tech Computer Center.

What is required is a script that runs periodically to discover which TCC accounts have Web homepages, and generate a page in the **www.nmt.edu/tcc** structure that has the standard appearance for pages in that structure.

See the online version of this manual<sup>1</sup>, or the print version<sup>2</sup>.

## 2. Operation

---

The job of the `homelist.py` script is pretty simple and self-contained. It uses the LDAP server to generate a list of current user accounts, and then looks in each account to see if there is a publicly accessible homepage. It then writes a list of those homepages (with links to the pages themselves) to the

---

<sup>1</sup> <http://www.nmt.edu/tcc/projects/homelist/>

<sup>2</sup> <http://www.nmt.edu/tcc/projects/homelist/homelist.pdf>

standard output stream. Redirecting this stream to the appropriate location will be taken care by the **crontab** entry that runs the script periodically.

Because it has standard library modules for accessing LDAP and file systems, as well as generating XHTML content, Python is a good choice for an implementation language.

At the moment, there is no easy way to generate XHTML pages that use the PyStyler<sup>3</sup> template that enforces the standard TCC web page style, other than to generate a PyStyler .g file that is part of the PyStyler structure.

To simplify the division of labor, the `homelist.py` script will generate only the content of the page's XHTML `<body>` element, and a static .g page and its .html equivalent will use a server-side include to import that generated content.

The XHTML content generated will have this form:

```
<table border='3' cellpadding='3'>
  <tr>
    <th>User name</th>
    <th>Homepage</th>
  </tr>
  <tr>
    <td>gecos</td>
    <td>
      <tt><a href='url'>url</a></tt>
    </td>
  </tr>
  ...
</table>
<p>
  This page is generated daily by a script. For documentation, see
  <a href='http://www.nmt.edu/tcc/projects/homelist/'
  ><citetitle><tt>homelist.py</tt>:
  A script to generate a list of user homepages</citetitle>
  </a>.
</p>
<p>
  Last updated YYYY-MM-DD HH:MM.
</p>
```

where the `<tr>` element is repeated for each user's **gecos** and the **url** of their home page.

The displayed list will be sorted by account name. The author would prefer to sort by last name and then first name, but unfortunately the **gecos** field is the entire name as "First Last", and there is no obvious way to discern where the last name begins (consider surnames like "de la O", "Ortiz y Vega", and "ter Horst").

## 2.1. Finding user homepages

A user will be considered to have a homepage if they have a world-readable directory named either `public_html` or the older equivalent `www`, and that directory contains a world-readable file named either `index.html` or the older equivalent `homepage.html`.

<sup>3</sup> <http://www.nmt.edu/tcc/help/pubs/pystyler>

We could just ask the HTTP server to serve us the page with the appropriate URL, but this has the undesirable side effect of causing a page fetch that will show up in our Web statistics on user page fetches. Also, why bother the HTTP server when we can just look for the user files directly?

## 2.2. LDAP considerations

The TCC's LDAP server will be used as the authority for the current set of user account names. The name of the LDAP server, and the DN (distinguished name) of the account root, will be hardcoded into the script. For more information about LDAP, see the document on the proposed *TCC Secure LDAP editing facility*<sup>4</sup>.

The server address is **ldaps://ldap0.nmt.edu:636**, and the base DN for accounts is **ou=accounts,dc=tcc,dc=nmt,dc=edu**. We need only two attributes:

- The **uid** attribute, which is the account name.
- The **gecos** attribute, which is the person's real name.

The first version of this script caused a problem because it tried to look for user homepages for all the "machine accounts," special accounts created for each user area workstation. Because such accounts have their home directories specified as `/dev/null`, the script caused many failures of the automounter to bind to `/dev/null`. We get around that by ignoring accounts whose **uid** attributes end with "\$".

## 2.3. crontab setup

At this writing, the homepage is periodically rebuilt from user **john**'s user crontab on the **infohost** server. It must be run on this server in order to have read-write access to the `/u/www/` filesystem.

To install a cron job to run this script periodically:

1. Login to **infohost**.
2. Display the old user crontab using this command:

```
crontab -l
```

3. Edit the user crontab using this command:

```
crontab -e
```

This brings up the user's editor of choice.

4. Make sure there is a line in the table with this format:

```
0 4 * * * (cd /u/www/docs/tcc;rm hompagelist.html;\n nice python homelist.py >hompagelist.html)
```

This runs the script every morning at 4am, niced down.

Here is the `homelist.g` file that PyStyler uses to build the `homelist.html` page containing the list of user homepages:

```
<head>\n  <author>John Shipman, <tt>tcc-doc@nmt.edu</tt></author>\n  <updated date="$Date: 2007/12/13 23:16:06 $">
```

<sup>4</sup> <http://www.nmt.edu/tcc/projects/ldap-edit/>

```
</head>

<p>Here is a list of TCC users who have put up Web homepages.
It is sorted by login name; to find the login name for a
person, see <r id="users">.

<!--#include virtual="homepagelist.html"-->
```

This page uses a server-side include to insert the file generated by the **make homelist** target.

## 3. Program source

This section describes the actual code of the `homelist.py` script, in literate programming style. For a definition of literate programming, as well as the tool used to extract the code from the document source, see *A source extractor for lightweight literate programming*<sup>5</sup>.

### 3.1. The prologue

Since Python scripts have the main at the end, you may wish to jump right to Section 3.10, “The main program” (p. 12).

The script starts out with a line to make it self-executing according to the usual Unix conventions. Next comes a brief comment pointing back to this document.

```
homelist.py
#!/usr/local/bin/python
#=====
# homelist.py: Generate a list of user homepages.
# For documentation, see:
# http://www.nmt.edu/tcc/projects/homelist/
```

Next we document the overall intended function of the script. For more information on intended functions and the Cleanroom programming methodology, see the author's Cleanroom pages<sup>6</sup>.

```
homelist.py
#-----
# Overall intended function:
# [ sys.stdout += an XHTML table containing a list of user
#   names and homepage links for all users in the TCC's LDAP
#   server who have a homepage ]
#-----
```

### 3.2. Imported modules

Next comes a few **import** statements to bring in the various Python modules we need.

- Since we're using Python generators, a relatively new feature, we have to import generators as a “future” feature before doing any other imports.

<sup>5</sup> <http://www.nmt.edu/tcc/help/lang/python/examples/litsource/>

<sup>6</sup> <http://www.nmt.edu/~shipman/soft/clean/>

```
#=====
# Imports
#-----
from __future__ import generators    # Allow generators
```

- The **sys** module contains the standard I/O streams such as **sys.stdout**, the standard output stream.

```
import sys
```

- The Python **ldap** module handles communication with the LDAP server. For more information, see *LDAP client API for Python*<sup>7</sup>.

```
import ldap
```

- The **os** module allows us to access Posix file systems so we can check for readable homepages. The related **stat** module is used for interpreting status values from a directory entry.

```
import os
import stat
```

- The Python **time** module is needed so we can timestamp the output.

```
import time
```

- The **xmlcreate.py** module is a package for XML file generation. Its interface is described in *Python and the XML Document Object Model (DOM)*<sup>8</sup>.

```
import xmlcreate as xc
```

### 3.3. Manifest constants

These constants, used throughout the program, are placed at the top for easy modification.

#### 3.3.1. UID\_ATTR

The name of the LDAP attribute that contains the user's account name.

```
#=====
# Manifest constants
#-----
UID_ATTR = "uid"
```

<sup>7</sup> <http://python-ldap.sourceforge.net/>

<sup>8</sup> <http://www.nmt.edu/tcc/help/pubs/pyxml/>

### 3.3.2. GECOS\_ATTR

The name of the LDAP attribute that contain's the user's full name (first plus last).

homelist.py

```
GECOS_ATTR = "gecos"
```

### 3.3.3. LDAP\_SERVER

The URL for our LDAP server.

homelist.py

```
LDAP_SERVER = "ldaps://ldap0.nmt.edu:636"
```

### 3.3.4. ACCOUNTS\_DN

The distinguished name (DN) of the LDAP root node for user accounts.

homelist.py

```
ACCOUNTS_DN = "ou=accounts,dc=tcc,dc=nmt,dc=edu"
```

### 3.3.5. WEB\_DIR

Name of the top-level subdirectory in a user account where all their Web pages must reside.

homelist.py

```
WEB_DIR = "public_html"
```

## 3.4. createTable(): Set up the XHTML table

This function sets up an empty XHTML table as a child of the **doc** node containing the XHTML document fragment. For the overall structure of the XHTML, see Section 2, "Operation" (p. 1).

homelist.py

```
# - - -   c r e a t e T a b l e   - - -

def createTable ( doc ):
    """Create an empty table.

    [ doc is a DOM DocumentFragment node ->
      doc := doc containing an empty XHTML table
      return that XHTML table as an Element node ]
    """
    #-- 1 --
    # [ doc := doc with a table element added
    #   result := that table element ]
    table = xc.Element ( doc, 'table', border='3', cellpadding='3' )

    #-- 2 --
    # [ table := table with its heading row added ]
    headRow = xc.Element ( table, 'tr' )

    head1 = xc.Element ( headRow, 'th' )
    xc.Text ( head1, 'User name' )
```

```

head2 = xc.Element ( headRow, 'th' )
xc.Text ( head2, 'Homepage' )

#-- 3 --
return table

```

### 3.5. scanAccounts(): Find all homepages

This routine takes the **table** node, containing the XHTML table we're building, and scans through all user accounts in the LDAP server, adding rows to the table for users who have homepages.

homelist.py

```

# - - -   s c a n A c c o u n t s   - - -

def scanAccounts ( table ):
    """Scan through all the user accounts, looking for homepages.

    [ table is a DOM Element node ->
      table := table with XHTML rows added representing a list of
        user names and homepage links for all users in the TCC's
        LDAP server who have a homepage ]
    """

```

First we call **generateAccounts()**, which iterates through the LDAP accounts list and returns an unordered list of (**uid**, **gecos**) 2-tuples, one for each user account.

homelist.py

```

#-- 1 --
# [ userList := a list of (uid, gecos) 2-tuples, one for
#   each active account in the LDAP server LDAP_SERVER ]
userList = generateAccounts()

```

Next, we sort the list by **uid**:

homelist.py

```

#-- 2 --
userList.sort()

```

Finally, we work through this list, and for each **uid** that has a homepage, we add another row to the table.

homelist.py

```

#-- 3 --
for uid, gecos in userList:
    #-- 3 body --
    # [ (uid is an account name) and
    #   (gecos is that account's personal name) ->
    #   if (uid doesn't end in '$') and
    #   (account uid has a homepage) ->
    #   table += an XHTML row added for account=uid
    #           and name=gecos ]
    if ( ( uid[-1] != '$' ) and
        hasHomepage ( uid ) ):
        addRow ( table, uid, gecos )

```

### 3.6. generateAccounts(): Search LDAP for accounts

The purpose of this function is to query the LDAP server and generate a list of (**uid**, **gecos**) pairs for each user account.

homelist.py

```
# - - -   g e n e r a t e   A c c o u n t s   - - -  
  
def generateAccounts():  
    """Query LDAP for all user accounts.  
  
    [ LDAP_SERVER names TCC's LDAP server and that server is up ->  
      return a list of (uid, gecos) tuples for each user  
      account, in no particular order ]  
    """
```

First, we bind to the LDAP server. Anonymous binding is sufficient to get the account information we want. See Section 3.3.3, “LDAP\_SERVER” (p. 6).

homelist.py

```
#-- 1 --  
# [ LDAP_SERVER names an accessible LDAP server ->  
#   anon := an LDAP object representing an anonymous bind  
#         to LDAP_SERVER ]  
try:  
    anon = ldap.initialize ( LDAP_SERVER )  
    anon.bind ( "", "", ldap.AUTH_SIMPLE )  
except ldap.LDAP_Error, detail:  
    print "*** Error, couldn't bind to the LDAP server."  
    sys.exit(1)
```

Next, we use the LDAP module's search function to search first-level children of the node specified by **ACCOUNTS\_DN**. The **filterstr** argument filters out entries that have no **uid** attribute. The **attrlist** argument requests that only the attributes named in the list are returned. See Section 3.3.4, “ACCOUNTS\_DN” (p. 6), Section 3.3.1, “UID\_ATTR” (p. 5), and Section 3.3.2, “GECOS\_ATTR” (p. 6).

homelist.py

```
#-- 2 --  
# [ anon is a bound LDAP object ->  
#   ldapResult := a list of LDAP result tuples for  
#                 a search of the immediate children of ACCOUNTS_DN that  
#                 have a UID_ATTR attribute, containing the UID_ATTR and  
#                 GECOS_ATTR attributes ]  
ldapResult = anon.search_s ( ACCOUNTS_DN, ldap.SCOPE_ONELEVEL,  
                             filterstr="(s=*)" % UID_ATTR,  
                             attrlist=[UID_ATTR, GECOS_ATTR] )
```

The structure of **ldapResult** is described by this excerpt from *LDAP programming with Python*:

Each result tuple is of the form (**dn**, **attrs**), where **dn** is a string containing the DN (distinguished name) of the entry, and **attrs** is a dictionary containing the attributes associated with the entry. The keys of **attrs** are strings, and the associated values are lists of strings.

So to produce the return value, we pull out the second element of each tuple in **ldapResult** to get the attribute map, then form a tuple from the first element of the list for each attribute we want.

There's one more complication: some LDAP entries don't have a GECOS field. In that case we just skip that entry.

homelist.py

```
#-- 3 --
# [ ldapResult is an LDAP-search-format result list ->
#   resultList := list of (uid, gecos) tuples made from
#               ldapResult ]
resultList = []
for dn, attrs in ldapResult:
    uid = attrs[UID_ATTR][0]
    try:
        gecos = attrs[GECOS_ATTR][0]
        resultList.append ( (uid, gecos) )
    except KeyError:
        pass

#-- 4 --
return resultList
```

### 3.7. hasHomepage ( ) : Test for the presence of a homepage

This function takes an account name and checks to see if the user has a top-level directory of the correct name (see Section 3.3.5, "WEB\_DIR" (p. 6)) that is world-readable and world-executable. (A former version of this program looked for a variety of homepage names, but the current Apache configuration will show a listing of the directory if there are no files whose names are considered homepages, so there is no point in looking at the contents of the directory.)

homelist.py

```
# - - -   h a s H o m e p a g e   - - -

def hasHomepage ( uid ):
    """Does this user have an HTML directory?

    [ uid is a string ->
      if uid is the account name of a user who has a
      homepage directory ->
        return 1
      else ->
        return 0 ]
    """
```

For a user account *N*, their home directory is at path `"/u/N"`, and their HTML directory must be a first-level directory whose name is given by Section 3.3.5, "WEB\_DIR" (p. 6).

homelist.py

```
#-- 1 --
# [ webPath := absolute path to the HTML directory for user
#           uid ]
webPath = "/u/%s/%s/" % (uid, WEB_DIR)

#-- 2 --
# [ if webPath names an existing path ->
#   mode := permissions word (mode bits) for webPath
#   else -> return 0 ]
```

```

try:
    statusTuple = os.stat ( webPath )
    mode = statusTuple [ stat.ST_MODE ]
except OSError:
    return 0

#-- 3 --
# [ if mode has world read and world execute bits both set ->
#     return 1
#   else ->
#     return 0 ]
return bool ( (mode & stat.S_IROTH) and
              (mode & stat.S_IXOTH) )

```

### 3.8. addRow() : Add a row to the result table

This function adds a row to the table containing the given **uid** and **gecos** fields.

homelist.py

```

# - - -   a d d R o w   - - -

def addRow ( table, uid, gecos ):
    """Add a result table row.

    [ (table is an XHTML table as a DOM Element node) and
      (uid is an account name) and
      (gecos is that accountholder's personal name) ->
      table := table with a new row added displaying gecos
              and the URL to uid as a link ]

    """

```

First we add the **tr** element with two **td** children:

homelist.py

```

#-- 1 --
# [ table := table with a new tr element added containing
#       two td children
#   tr   := that tr element
#   td1  := that first td child
#   td2  := that second td child ]
tr = xc.Element ( table, "tr" )
td1 = xc.Element ( tr, "td" )
td2 = xc.Element ( tr, "td" )

```

In the first cell we put the personal name:

homelist.py

```

#-- 2 --
# [ td1 := td1 with a new text element added containing gecos ]
xc.Text ( td1, gecos )

```

Finally, in the second cell we put the URL both as a link and as link text.

homelist.py

```

#-- 3 --
# [ td2 := td2 with a new tt element added containing a

```

```

#           link to the homepage url for user (uid) with the
#           url as the link text ]
url = "http://www.nmt.edu/~%s/" % uid
tt2 = xc.Element ( td2, "tt" )
a    = xc.Element ( tt2, "a", href=url )
xc.Text ( a, url )

```

### 3.9. addEpilogue(): Add end comments

For the XHTML generated after the end of the main table, see Section 2, "Operation" (p. 1).

homelist.py

```

# - - -   a d d E p i l o g u e   - - -

def addEpilogue ( doc ):
    """Add the end comments after the main table.

    [ doc is a Document object ->
      doc := doc with end comments added ]
    """

    #-- 1 --
    # [ doc := doc with a new paragraph element added
    #   p1 := that paragraph element ]
    p1 = xc.Element ( doc, "p" )
    xc.Text ( p1, "This page is generated daily by a script. "
              "For documentation, see\n" )

    #-- 2 --
    # [ p1 := p1 with a link to this document added with its
    #       title as the link text ]
    a = xc.Element ( p1, "a",
                    href="http://www.nmt.edu/tcc/projects/homelist/" )
    cite = xc.Element ( a, "citetitle" )
    tt    = xc.Element ( cite, "tt" )
    xc.Text ( tt, "homelist.py" )
    xc.Text ( cite, ":\nA script to generate a list of user homepages" )
    xc.Text ( p1, "." )

    #-- 3 --
    # [ doc := doc with a new paragraph element added, containing
    #       the date timestamp
    #   p2 := that paragraph ]
    p2 = xc.Element ( doc, "p" )
    xc.Text ( p2, "Last updated " )
    xc.Text ( p2, time.strftime ( "%Y-%m-%d %H:%M", time.gmtime() ) )
    xc.Text ( p2, " UT." )

```

### 3.10. The main program

In general, there are two main phases to the program. In the first phase, we build up a DOM tree representation of the XML content we'll be writing to the output. The second phase just consists of writing that tree to the standard output stream as XML.

We use the DOM's **DocumentFragment** type, instead of a full **Document**, in order to avoid getting a spurious `<?xml ...?>` processing instruction at the start of the output.

homelist.py

```
# - - - - -   m a i n   - - - - -

#-- 1 --
# [ doc      := a new, empty DOM DocumentFragment ]
doc = xc.DocumentFragment()

#-- 2 --
# [ doc      := doc containing an XHTML table with heading row
#   table    := that XHTML table as an Element node ]
table = createTable ( doc )

#-- 3 --
# [ table    := table with XHTML rows added representing a list of
#   user names and homepage links for all users in the TCC's
#   LDAP server who have a homepage ]
scanAccounts ( table )

#-- 4 --
# [ doc      := doc with end annotation added ]
addEpilogue ( doc )

#-- 5 --
# [ sys.stdout += doc represented as XHTML ]
doc.write()
```