```python
#!/usr/bin/env python
# mkapachepw.py
# Generate Apache-Compatible Password And Group Files
# From Unix System Passwords And Groups.
#
# Copyright (c) 2005 TundraWare Inc.  All Rights Reserved.
# For Updates See:  http://www.tundraware.com/Software/mkapachepw

# Program Information

PROGNAME = "mkapachepw"
RCSID = "$Id: mkapachepw.py,v 1.121 2005/04/12 07:10:35 root Exp $"
VERSION = RCSID.split()[2]

# Copyright Information

CPRT         = "(c)"
DATE         = "2005"
OWNER        = "TundraWare Inc."
RIGHTS       = "All Rights Reserved"
COPYRIGHT    = "Copyright %s %s %s  %s. " % (CPRT, DATE, OWNER, RIGHTS)


#----------------------------------------------------------#
#            Variables User Might Change                   #
#----------------------------------------------------------#

BOGUSID  = 100000               # Fake GID/UID used when external files
GRFILE   = "./.htgroups"        # Group output file
USRFILE  = "./.htusers"         # User output file
STARTUID = 100                  # User IDs below this ignored
STARTGID = 100                  # Group IDS below this ignored



#------------------- Nothing Below Here Should Need Changing ------------------#


#----------------------------------------------------------#
#                       Imports                            #
#----------------------------------------------------------#

import getopt
import grp
import os
import pwd
from socket import getfqdn
import sys
import time


#----------------------------------------------------------#
#                 Aliases & Redefinitions                  #
#----------------------------------------------------------#



#----------------------------------------------------------#
#                 Constants & Literals                     #
#----------------------------------------------------------#



#####
# Constants
#####



#####
# Literals
#####

CMDLINE   = "# Command Line: %s\n" % " ".join(sys.argv)
TIMESTAMP = "# Created By %s %s On %s At %s\n" % (PROGNAME, VERSION, getfqdn(), time.asctime())

GID       = 'GID'
```

```
GROUP     = 'Group'
UID       = 'UID'
USER      = 'User'


#----------------------------------------------------------#
#              Prompts, & Application Strings              #
#----------------------------------------------------------#


#####
# Error And Warning Messages
#####

eABORT          = "Aborting ..."
eERROR          = "ERROR"
eFILEOPEN       = "Cannot Open File '%s'."
eINVALIDID      = "'%s' Is An Invalid %s ID."
eINVALIDNAME    = "'%s' Is An Invalid %s Name."
eINVALIDSTART   = "Invalid Starting %s, '%s' - Must Be An Integer Value."
eNOPREFIX       = "'%s' Must Be Prefixed With '+' or '-' To Indicate Desired Action."

wCOLLIDE        = "'%s' Entry In %s Conflicts With Entry Already In %s List."
wOVERWRITE      = "Overwriting..."
wWARNING        = "WARNING"


#####
# Usage Prompts
#####

uTable = [PROGNAME + " " + VERSION + " - %s\n" % COPYRIGHT,
          "usage:  " + PROGNAME + " [-sGUguIicqOohv]",
          "   where,",
          "          -s       do not process system password/group files (default: process these files)",
          "          -G       list of groups to include (+group | +GID) or exclude (-group | -GID) (default: none)",
          "          -U       list of users to include (+user | + UID) or exclude (-user | -UID) (default: none)",
          "          -g #     smallest GID to include in output (default: 100)",
          "          -u #     smallest UID to include in output (default: 100)",
          "          -I file  include file containing other group information (default: none)",
          "          -i file  include file containing other user information (default: none)",
          "          -c       do not permit entries to be overwritten (default: allow - only warn)",
          "          -q       quiet mode - suppresses warning messages",
          "          -O file  group file name, - selects stdout (default: ./.htgroups)",
          "          -o file  user file name, - selects stdout  (default: ./.htusers)",
          "          -h       print this help information",
          "          -v       print detailed version information",
          ]


#----------------------------------------------------------#
#          Global Variables & Data Structures              #
#----------------------------------------------------------#

enumerated         = []          # Place to store command line in/exclude enumerations
includes           = []          # Place to store names of files to include

groups             = {}          # Place to store group information
users              = {}          # Place to store user information

ALLOWCOLLISIONS    = True        # Allow entries to overwrite each other (with warning)
QUIET              = False       # Suppress display of warning messages
SYSFILES           = True        # Flag to enable/disable inclusion of system group/pw


#--------------------------- Code Begins Here ---------------------------------#


#----------------------------------------------------------#
#             Object Base Class Definitions                #
#----------------------------------------------------------#



```

```
#----------------------------------------------------------#
#              Supporting Function Definitions             #
#----------------------------------------------------------#


#####
# Print An Error Message
#####

def ErrorMsg(emsg, Warning=False, Action=eABORT):

    if Warning:
        if QUIET:     # Quiet mode suppresses warning messages
            return
        prompt = wWARNING
    else:
        prompt = eERROR

    print PROGNAME + " " + VERSION + " " + prompt + ": " + emsg + " " + Action

# End of 'ErrorMsg()'


#####
# Print Usage Information
#####

def Usage():
    for line in uTable:
        print line

# End of 'Usage()'


#####
# Process An Enumerated List Of Groups/Users To Include Or Exclude.
#
# The 'items' argument must be a string with the names or numbers to
# process, with a '-' or '+' prepended to indicate Delete or Add,
# respectively.
#####

def ProcessEnumeratedList(items, lookup, name):

    if name == GROUP:
        master = groups
    else:
        master = users

    for item in items.split():
        orig = item

        # Verify argument is in correct format and determine type of
        # operation desired.

        if item[0] == '-':
            additem = False

        elif item[0] == '+':
            additem = True

        else:
            ErrorMsg(eNOPREFIX % item)
            sys.exit(2)

        item = item[1:]     # We just need the item Name/ID portion

        # Convert GIDs and UIDs to names first
        try:
            item = int(item)

            # Handle the case where the ID does not exist
            try:
                item = lookup(item)[0]

            except:
                ErrorMsg(eINVALIDID % (orig[1:], name))
```

```
                sys.exit(2)

        # If not, assume it is a name and look it up
        except ValueError:

            # Make sure it even exists

            if item not in master:
                ErrorMsg(eINVALIDNAME % (item, name))
                sys.exit(2)

        # Do the actual in/exclusion

        # Include
        if additem:
            master[item][2] = True   # Mark entry as protected

        # Exclude
        else:
            del master[item]

# End of 'ProcessEnumeratedList()'


#####
# Read A File Into A Local List, Stripping Newlines, And
# Suppressing Blank Lines
#####

def ReadFile(filename):

    temp = []
    try:
        f = open(filename)
        for l in f.readlines():
            if l[-1] == '\n':            # Get rid of trailing newlines
                l = l[:-1]
            l = l.split('#')[0].strip()  # Get rid of comments

            if l:                        # Add any non-blank lines
                name, members = l.split(':')
                members = members.split()
                temp.append([name, members])
        f.close()

    except:
        ErrorMsg(eFILEOPEN % filename)
        sys.exit(1)

    return temp

# End of 'ReadFile()'


#----------------------------------------------------------#
#                   Program Entry Point                    #
#----------------------------------------------------------#


#####
# Command line processing - Process any options set in the
# environment first, and then those given on the command line
#####

OPTIONS = sys.argv[1:]
envopt = os.getenv(PROGNAME.upper())
if envopt:
    OPTIONS = envopt.split() + OPTIONS

try:
    opts, args = getopt.getopt(OPTIONS, '-sG:U:g:u:I:i:cqO:o:hv')
except getopt.GetoptError:
    Usage()
    sys.exit(1)

# This command line accepts no args
if args:
```

```
    Usage()
    sys.exit(1)


for opt, val in opts:
    if opt == "-s":
        SYSFILES = False
    if opt == "-G":
        enumerated.append([val, grp.getgrgid, GROUP])
    if opt == "-U":
        enumerated.append([val, pwd.getpwuid, USER])
    if opt == "-g":
        try:
            STARTGID=int(val)
        except:
            ErrorMsg(eINVALIDSTART % (GID, val))
            sys.exit(1)
    if opt == "-u":
        try:
            STARTUID=int(val)
        except:
            ErrorMsg(eINVALIDSTART % (UID, val))
            sys.exit(1)
    if opt == "-I":
        includes.append([val, groups])
    if opt == "-i":
        includes.append([val, users])
    if opt == "-c":
        ALLOWCOLLISIONS = False
    if opt == "-q":
        QUIET = True
    if opt == "-O":
        GRFILE = val
    if opt == "-o":
        USRFILE = val
    if opt == "-h":
        Usage()
        sys.exit(0)
    if opt == "-v":
        print RCSID
        sys.exit(0)

#####
# Build List Of System Groups And Users
#####


# Can be suppressed with the -s command line argument

if SYSFILES:

    Protected = False

    #####
    # Build List Of Groups
    #####

    for group in grp.getgrall():

        gname, gpw, gid, gmembers = group[:4]

        groups[gname] = [gid, [], Protected]
        for member in gmembers:
            groups[gname][1].append(member)

    #####
    # Build A List Of Users
    #####

    for user in pwd.getpwall():

        uname, pw, uid, gid = user[:4]
        gname = grp.getgrgid(gid)[0]

        users[uname] = [uid, pw, Protected]

        if uname not in groups[gname][1]:
```

```
            groups[gname][1].append(uname)


#####
# Process Included Files
#####

for include in includes:

    # Read the file into a temporary list
    filename, db = include[:]
    temp = ReadFile(filename)

    # Add each entry to the appropriate in-memory database
    for entry in temp:

        # Group entries have a list of members
        if db == groups:
            members = entry[1]
            name = GROUP

        # User entries have a single password
        else:
            members = entry[1][0]
            name = USER

        # See if this entry will overwrite an existing one
        # If it will, warn if collisions are permitted
        # Error out otherwise

        if entry[0] in db:

            if ALLOWCOLLISIONS:
                ErrorMsg(wCOLLIDE % (entry[0], filename, name), Warning=True, Action=wOVERWRITE)
            else:
                ErrorMsg(wCOLLIDE % (entry[0], filename, name))
                sys.exit(4)

        db[entry[0]] = [BOGUSID, members, False]


#####
# Process Any Enumerated Inclusions/Exclusions
#####

for enum in enumerated:
    ProcessEnumeratedList(*enum)


#####
# Write Out The Files
#####

# Files Should Be Read-Only

os.umask(0377)

# Group File

try:
    if GRFILE == '-':
        grfile = sys.stdout
    else:
        grfile = open(GRFILE, "w")
except:
    ErrorMsg(eFILEOPEN % GRFILE)
    sys.exit(3)

grfile.write(TIMESTAMP)
grfile.write(CMDLINE)

# Write out groups if they are either protected or >= specified starting ID

gnames = groups.keys()
gnames.sort()
for gname in gnames:
    if (groups[gname][2]) or (groups[gname][0] >= STARTGID):
```

```
        grfile.write("%s: %s\n" % (gname, " ".join(groups[gname][1])))

grfile.close()


# Password File

try:
    if USRFILE == '-':
        pwfile = sys.stdout
    else:
        pwfile = open(USRFILE, "w")
except:
    ErrorMsg(eFILEOPEN % USRFILE)
    sys.exit(3)

pwfile.write(TIMESTAMP)
pwfile.write(CMDLINE)

# Write out users if they are either protected or >= specified starting ID
# Unless explicitly protected, any account that has '*' as a password
# (thus indicating it does not support login), will be suppressed.

unames = users.keys()
unames.sort()
for uname in unames:
    if (users[uname][2]) or ((users[uname][0] >= STARTUID) and (users[uname][1] != '*')):
        pwfile.write("%s:%s\n" % (uname, users[uname][1]))

pwfile.close()
```