RANDMUSIC - COPY RANDOM SONGS TO AN MP3 PLAYER

This program was not specifically written for a class that I teach, but I may use it in a future class. This is a fairly simple program that demonstrates working with drives, directories and files. The usage of a number of Python modules is demonstrated. These include: string, random, sys, os, os.path, shutil and getopt. Additionally, getoutput is used in the Unix environment and pywin32 is used when used on a Windows based computer. This program does not use object-oriented programming, so it is suitable for study by beginning students.

The motivation for writing this program was that periodically I want to transfer a random sampling of music to a basic mp3 player. I used to just transfer music files until it is full, but that is tedious and time consuming; I end up with some of the same songs; and the last copy fails when it is full.

USAGE

First, generate a list of songs that might be transferred to the player. I then edited the list to remove some songs (file names) I don't want to listen to. (My daughter added some songs that I don't like – such is life.) Working with a list of files from a file also allows for some customization of which songs end up on the MP3 player (example of that below). You need to want to pay attention to the absolute path names of the files. My music files all reside in a directory under my home directory named Music and in the root directory of the player is a directory named *Music*, which is the destination directory for the music. So, I generate my list so that each line begins with *Music/*...:

```
cd
find Music -type f > allmp3s
vi allmp3s
```

Next, plug in the MP3 player and note its mount point. For this discussion, I'll use a mount point of */mnt/*player.
To fill the player with a new random sampling of songs, we can use the –fill option:

```
$ randmusic.py allmp3s /mnt/player --fill
```

NOTE:
The **randmusic.py** program does not assume that the device should be completely filled. With no optional arguments (other than *source* and *destination*), will replace the song on the player filling approximately the same amount of space as was initially on the player.

If we know that we need to reserve a certain amount of space on the player for something we want to add to it later, we can use the *–add* to fill the player to a specific level. The used space on the player after the *–add* option is used is the current space used plus the value supplied with the *–add* option.

```
$ randmusic.py allmp3s /mnt/player --fill
```

Sometimes, you may not wish to remove any files on the player, but just want to add additional files to it. This is when the *–keep* option is used. The *–keep* option comes in handy to load certain songs and then fill the remaining space on the device with a random sampling of songs.

```
$ randmusic.py favorites /mnt/player
$ randmusic.py allmp3s /mnt/player --keep --fill
```

AUTO GENERATED DOCUMENTATION

File: randmusic.py
Usage:
$ randmusic.py src dest [(-a | –add) x] [-f | –fill] [-k | –keep]
or
$ randmusic.py –help
Copy a random set of files, which have a collective size just under the capacity of a device such as a mp3 player.
Author: Tim Bower
Kansas State University at Salina
Apache Open Source License, V2.0

`randmusic.`**`checkFile`***(filename)*

> Check that a file exists, is readable and return it's size. If there is a problem with reading the file, it returns zero. Note, this could use os.access() and os.path.getsize() instead of os.stat(). With stat, we get both at once plus a chance to learn how to use it.

`randmusic.`**`randfiles`***(src, fsSize, dest=None)*

From src (a file with a list of files), randomly pick as many of the files as will fit in the specified file system size.

`randmusic.`**`calc_used_file_size`***(dest)*

Like 'du -ks dest', determine the disk usage by all the files currently under the dest directory. This size plus the available space on the drive is total available space for files.

`randmusic.`**`smart_copy`***(files, dest, keep=False)*

Copy the files in files list to dest directory, but do it smart (faster). Rather than deleting all the files under dest and then copying all the files, delete only those files under dest not in files, and then copy only the files not already under dest. Finally, it removes some empty directories.

`randmusic.`**`get_disk_space`***(dest)*

Determine the capacity and available space on the device. This code has platform dependent components and currently supports posix (Unix) compatible systems and Windows.

`randmusic.`**`up_date_device`**(*src, dest, add=0, fill=False, keep=False)*

Parameters:

- src – file
- dest – directory
- add – (int) kb max disk usage on device. If add == 0, then go off what the current disk usage is.
- fill – boolean for if we want the device as full as possible
- keep – boolean, should files on device be retained

The driving function for updating the mp3 player with new files.

`randmusic.`**`usage()`**

Prints correct program usage information.

randmusic.py CODE

```
#!/usr/bin/env python

"""
**File:** randmusic.py

Usage:

$ randmusic.py src dest [(-a | --add) x] [-f | --fill] [-k |
--keep]

or

$ randmusic.py --help

Copy a random set of files, which have a collective size just
under
the capacity of a device such as a mp3 player.


**Author:** Tim Bower

Kansas State University at Salina

Apache Open Source License, V2.0
"""

# For details, see documentation on web page.
# http://www.sal.ksu.edu/faculty/tim/
# Copyright 2009 Tim Bower
#
# This program is licensed as Open Source Software using the
Apache License,
# Version 2.0 (the "License"); you may not use this file except
in compliance
# with the License. You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
```

```
# You are free to use, copy, distribute, and display the work,
and to make
# derivative works. If you do, you must give the original
author credit.
#
# Version 1, Fall 2008 -- just make a random list of files (of
correct size),
#    but deleting old files and copying new ones must be done
manually.
# Version 2, April 2009 -- use random.shuffle() - a simpler
design.
# Version 3, May 2009 - don't just make a list to copy, but
remove
#    files not retained and do the copy.
# Version 4,    - mostly refactorying - split into more
functions for cleaner
#    hanlding of options, such as --keep; if files kept, don't
under fill when
#    files are already on the destination.  Try optparse
instead of getopt.


import string
import random
import sys
import os
import stat
from os.path import getsize, exists, join, dirname, isfile,
isdir, commonprefix
import shutil
import getopt

def checkFile( filename ):
    """
    Check that a file exists, is readable and return it's size.
    If there is a problem with reading the file, it returns
zero.
    Note, this could use os.access() and os.path.getsize()
instead of
```

```
    os.stat().  With stat, we get both at once plus a chance to
learn
    how to use it.
    """
    if not os.path.exists( filename ): return 0
    st = os.stat(filename)
    mode = st[stat.ST_MODE]
    if not mode & stat.S_IREAD: return 0 # same as stat.S_IRUSR
    return st[stat.ST_SIZE] >> 10 # convert file size to K

def randfiles(src, fsSize, dest = None):
    """
    From src (a file with a list of files), randomly pick as
many of
    the files as will fit in the specified file system size.
    """
    src_file = open(src, 'r')
    files = src_file.readlines()
    src_file.close()

    files = [string.strip(x) for x in files]
    random.shuffle(files)

    randfiles = []
    size = 0
    smallest = fsSize
    for file in files:
        if dest and exists(join(dest,file)):
            continue
        fsize = checkFile(file)
        if fsize == 0:
            sys.stderr.write("Could not find file " + file +
"\n")
            continue
        if fsize < smallest:
            smallest = fsize
        if size + fsize < fsSize:
            size += fsize
            randfiles.append(file)
        elif size + smallest > fsSize:
            # we are close enough now, no point continuing
            break
    #--
    if len(randfiles) == 0:
        print "No apparent capacity for new files on
destination"
    else:
        # sort them to put ones from same folder together.
        randfiles.sort()
    return randfiles

def calc_used_file_size(dest):
    """
    Like 'du -ks dest', determine the disk usage by all the
files
    currently under the dest directory.  This size plus the
available
    space on the drive is total available space for files.
    """
    size = 0
    for root, dirs, files in os.walk(dest):
        size += sum([getsize(join(root, name)) >> 10 for name
in files])
    return int(size)

def smart_copy(files, dest , keep = False):
    """
    Copy the files in files list to dest directory, but do it
smart
    (faster).  Rather than deleting all the files under dest
and then
    copying all the files, delete only those files under dest
not in files,
    and then copy only the files not already under dest.
Finally, it
    removes some empty directories.
    """

    keep_list = []
    copy_list = files[:]
    for file in files:
        dest_file = join(dest,file)
        if exists(dest_file):
            copy_list.remove(file)
            keep_list.append(dest_file)

    if not keep:
        empty_dirs = []
        print "\nDeleting the following files:"
        base = join(dest, commonprefix(files))
        for root, dirs, files in os.walk( base ):
            del_files = filter(lambda f: join(root,f) not in
keep_list,
                               files )
            for del_file in del_files:
                df = join(root, del_file)
                print df
                os.remove( df )

            if len(del_files) == len(files):
                # Possible empty directories, not all of these
will be
                # deleted. They may have other directories or
files may
                # get copied here.  Note, this test is more
inclusive than
                # testing os.listdir() because it may contain a
                # directory, which will also end up empty.
                empty_dirs.append(root)

    # Note: Since the mp3player uses a fat (dos) filesystem,
don't
    # worry about file permissions.
    if len(copy_list) > 0:
        print "\nCopying the following files:"
        for file in copy_list:
            print file
            dest_file = join(dest,file)
            base_dir = dirname(dest_file)
            if not exists(base_dir):
                os.makedirs(base_dir)
            shutil.copyfile( file, dest_file )
    else:
        print "No files added this time"

    if not keep:
        print "\nRemoving the following empty directories:"
        # reverse the list of possible empty directories so
that we check
        # and delete depth first.  Some may become empty when
        # sub-directories are deleted first.
        empty_dirs.reverse()
        for emptyd in empty_dirs:
            # check if still empty
            if len(os.listdir( emptyd )) == 0:
                print emptyd
                os.rmdir(emptyd)

def get_disk_space(dest):
    """
    Determine the capacity and available space on the device.
    This code has platform dependent components and currently
supports posix
    (Unix) compatible systems and Windows.
    """
    if os.name == 'posix':
        # get available space from df command
        from commands import getoutput
        try:
            out = getoutput('df -kP %s' % dest).split()[8:11]
            capacity = int(out[0])
            avail = int(out[2])
        except:
            print "Failed to get disk usage information"
            raise
    else:
        try:
            # need to test this...
            from pywin32 import GetDiskFreeSpace
            disk_tuple = GetDiskFreeSpace(dest)
            sectors_p_cluster = disk_tuple[0]
            bytes_p_sector = disk_tuple[1]
            free_clusters = disk_tuple[2]
            total_clusters = disk_tuple[3]
            kbytes_p_cluster = (sectors_p_cluster *
bytes_p_sector)>> 10
            capacity = total_clusters * kbytes_p_cluster
            avail = free_clusters * kbytes_p_cluster
        except:
            print "Failed to get disk usage information"
            raise
    return capacity, avail

def up_date_device(src, dest, add=0, fill=False, keep=False):
    """
    :param src: file
    :param dest: directory
    :param add: (int) kb max disk usage on device. If add == 0,
then go off
                what the current disk usage is.
    :param fill: boolean for if we want the device as full as
possible
    :param keep: boolean, should files on device be retained

    The driving function for updating the mp3 player with new
files.
    """
    if not (os.access(src, os.R_OK) and isfile(src)):
        print "%s can not be read as regular file" % src
        raise IOError

    if not isdir(dest):
        print "%s does not exist or is not a directory" % dest
        r = raw_input("Should we try to create a directory?
(y/n) [y]: ")
        if not r: r = 'y'
        if r[0].lower() == 'y':
            try:
                os.makedirs(dest)
            except:
                print "Failed to create directory %s: %s" %\
                    (dest, sys.exc_info()[1][1])
                raise IOError
    if not os.access(dest, os.W_OK | os.X_OK):
        print "You do not have permission to write files to %s"
% dest
        raise IOError

    print "Determining available space"
    disk_used = calc_used_file_size(dest)
    capacity, avail = get_disk_space(dest)
    if fill:
        # player does not want to be filled completely full
        fsize = 0.95*min(capacity, avail + disk_used)
    else:
        fsize = 0.95*min(capacity, add + disk_used)
    disk_to_check = None
    if keep:
        fsize -= disk_used
        disk_to_check = dest
    #--- end fsize calc
    # note fsize is in kb, but that's a big number, so report
it
    # in Mb.
    if fsize > 0:
        print "Will copy %f Mb of files" %
(float(fsize)/1024.0)
        print "Finding a new set of random files"
        files = randfiles(src, fsize, disk_to_check)
        if len(files) > 0 or not keep:
            smart_copy( files, dest, keep )
        else:
            print "No change to the device this time."
    else:
        print """Did not find any available disk space on %s.
        Check the disk and your command line arguments.""" %
dest
        usage()

def usage():
    """ Prints correct program usage information."""

    print """Usage:
    %s source destination
```

```
            [(-a | --add) n] [-f | --fill] [-k | --keep]

    Source: a file containing a list of available files.
    Destination: the path to where the files should be copied.
    (optional)
    --add n: n is the additional available space (in kb).
    --fill: Fill the device to capacity.
    --keep: Do not delete any files, just add to it.

    Defaults behavior is to remove files from the device, and
to fill
    the device to approximately it's current level.

    See the documentation.
    """ % sys.argv[0]

if __name__ == '__main__':
    argc = len(sys.argv)
    if argc > 2:
        src = sys.argv[1]
        dest = sys.argv[2]
        add = 0
        fill = False
        keep = False
        if argc > 3:
            try:
                opts, args = getopt.getopt(sys.argv[3:],
"-a:fk",
                        ["add", "fill", "keep"])
            except getopt.GetoptError, err:
                # print help information and exit:
                print str(err)
                # will print something like "option -x not
recognized"
                usage()
                sys.exit(1)
            for o, a in opts:
                if o in ("-a", "--add"):
                    add = int(a)
                if o in ("-f", "--fill"):
                    fill = True
            if o in ("-k", "--keep"):
                keep = True
    #--- end getopt --
    try:
        up_date_device(src, dest, add, fill, keep)
    except:
        print "Update Failed"
        raise
        sys.exit(1)
    print "Update of %s succeeded" % dest
else:
    usage()
    sys.exit(2)
```