```
import os
import pwd
import sys
import argparse
import hashlib
import subprocess
import re
#RUN WITH PYTHON 3

#For the given path, returns sha256 hash value of the file.
def file_hasher(filepath):
    return hashlib.sha256(open(filepath, 'rb').read()).hexdigest()

#For the given path, returns sha256 hash value of the directory.
def dir_hasher(filepath):
    dir_content = os.listdir(filepath)
    dir_hash = []
    for fdir in dir_content:
        if os.path.isdir(filepath + "/" + fdir):
            for hashval, direct in alldirectories.items():
                if direct == (filepath + "/" + fdir):
                    dir_hash.append(hashval)
        else:
            dir_hash.append(file_hasher(filepath + "/" + fdir))
    dir_hash.sort()
    direc_hash = ''.join(dir_hash)
    return hashlib.sha256(direc_hash.encode('utf-8')).hexdigest()
#Given hash value, [path] dictionary, returns the duplicate list.
def find_dups(d_list, regexx):
    rets = []
    for key,value in d_list.items():
        x = list(set(value))
        t = []
        for item in x:
            if(re.search(regexx,item.split("/")[-1])):
                t.append(item)
        #Dictionary format: {<hash_value> : [list_of_files_corresponds_to_hash_value]}
        #If list length is greater than one, then there are duplicates.
        if(len(t)>1):
            rets.append(t)
    return rets

#Executes the given command on the list of directories using subprocess.
def command_execute(command,list):
    if(command == 'p'):
        for i in list:
            for item in i:
                print(item)
            print(" ")
    else:
        for i in list:
            for item in i:
                cmd = command + " " + item.replace(" ","\ ")
                output = subprocess.check_output(cmd, shell=True)
                #output = subprocess.call(cmd, shell=True)
                print (output)

cwd = os.getcwd()       #Getting current working director

parser = argparse.ArgumentParser()
#The command
actions = parser.add_mutually_exclusive_group()
actions.add_argument(
```

```
    '-p', '--print', action='store_const', dest='action', const='p', default='p')
actions.add_argument(
    '-c', '--command', action='store', dest='action', type=str)
#The type
types = parser.add_mutually_exclusive_group()
types.add_argument(
    '-f', '--file', action='store_const', dest='type', const='f', default='f')
types.add_argument(
    '-d', '--directory', action='store_const', dest='type', const='d')
#List of directories
parser.add_argument("dirs", type=str,  default=[cwd], nargs='*')

args = parser.parse_args()
dirlist = args.dirs

#Regex pattern is initally empty, if first argument has quotes in it,
#First element is assigned to regex variable.
regex = ""
if dirlist[0][0] == '\"':
    regex = dirlist.pop(0)[1:-1]

if(len(dirlist)==0):
    dirlist.append(cwd)
#All files,directories and their hash values are stored in the below directories
#Format: {<hash_value> : [list_of_files_corresponds_to_hash_value]}
allfiles = {}
alldirectories = {}

#Walking and hashing all files and directories.
for fullpath in dirlist:
    for root, dirs, files in os.walk(fullpath, topdown=False):
        for fname in files:
            file_path = root + "/" + fname
            file_hash = file_hasher(file_path)
            if file_hash in allfiles:
                allfiles[file_hash].append(file_path)
            else:
                allfiles[file_hash] = [file_path]
        for dname in dirs:
            dir_path = root + "/" + dname
            dir_hash = dir_hasher(dir_path)
            if dir_hash in alldirectories:
                alldirectories[dir_hash].append(dir_path)
            else:
                alldirectories[dir_hash] = [dir_path]

#Control for file or directory, then executing commands.
if (args.type == 'f'):
    command_execute(args.action, find_dups(allfiles,regex))
else:
    command_execute(args.action, find_dups(alldirectories,regex))
```