CMPE230
Project-2
2016400039 – Irmak GÜZEY / 2016400297 - Hasan Basri BALABAN

INTRODUCTION

In this project, we are asked to implement a file utility program called **filelist** that will traverse directories and report path names of files that satisfy some search criteria. The program can be invoked as follows on the console:

filelist [options] [directory list]

The arguments in [ ] are optional. If they are not given, the default action will be carried out. If no directory list is given, the default will be the current directory. If there is no option arguments, pathnames of all files will be printed by the program. The available options are as follows:

-before datetime **:** Files that are last modified before a date or a datetime.

-after datetime **:** Files that are last modified after a date or a datetime.

-match <pattern> **:** File names that are matching a Python regular expression **<pattern>.**

-bigger <int> **:** Files that have sizes greater than or equal to **<int>** bytes. (**<int>** can also be given as kilobytes (K), megabytes (M) and gigabytes (G)).

-smaller <int> **:** Files that have sizes less than or equal to **<int>** bytes. (**<int>** can also be given as kilobytes (K), megabytes (M) and gigabytes (G)).

-delete **:** This option deletes the files that are in the given directory recursively.

-zip <zipfile> **:** This option packs the necessary files as a zip file.

-duplcont **:** Lists all the files that have the same content in sorted order by separating the duplicate sets of files with **"-----"** characters.

-duplname **:** Lists all the files that have the same name in sorted order by separating the duplicate sets of files with **"-----"** characters.

-stats **:** Prints traversal statistics at the end of files listing output. The statistics are as follows:

- Total number of files visited
- Total size of files visited in bytes
- Total number of files listed
- Total size of files listed in bytes

If **-duplcont** option is given, the following informations will be printed additionally:

- Total number of unique files listed
- Total size of unique files listed in bytes

If **-duplname** option is given, the following information will be printed additionally:

- Total number of files with unique names.

-nofilelist **:** No file listing will be printed.

Notes : →If multiple options are present, the files that satisfy conditions of all options will be returned. (option conditions are **AND**ed)
→Only one of **-duplcont** or **-duplname** can be given.

IMPLEMENTATION OF PROJECT

First of all, we used **argparse** parser in the project to parse the command-line options. An argparse object that named **parser** holds all the arguments that can be given as options and **args** object holds the parsed command-line command by parser object. An example of adding arguments to parser object is given below:

→parser.add_argument("doc_list", nargs="", help="to print the files in the given document list")*
→parser.add_argument("-zip", nargs="?", default="None", help="zips the filtered files")

After parsing the given command-line commands and holding all the given options in args object, we declare a list called **out** which will hold all the files that satisfy the given conditions and will be printed at the end of the program. The implementations of options are done as follows:

#DOC_LIST
This option traverses the given directory recursively and appends every file to out list. It uses an if block to perform this procedure. It checks if a directory is given. If given then traverse all the files inside that directory, if not given then traverse all the files inside the current directory. The implementation is as follows:

```
if args.doc_list:
        list = args.doc_list
        for argv_new in list:
                qlist = deque([argv_new])
                while qlist:
                        currentdir = qlist.popleft()
                        dircontents = os.listdir(currentdir)
                        for name in dircontents:
                                currentitem = currentdir + "/" + name
                                if os.path.isdir(currentitem):
                                        qlist.append(currentitem)
                                else:
                                        # print(currentitem)
                                out.append(currentitem)
else:
        argv_new = "."
        qlist = deque([argv_new])
        while qlist:
                currentdir = qlist.popleft()
                dircontents = os.listdir(currentdir)
                for name in dircontents:
                        currentitem = currentdir + "/" + name
                        if os.path.isdir(currentitem):
                                qlist.append(currentitem)
                        else:
                                #print(currentitem)
                        out.append(currentitem)
```

After #doc_list, program counts all the files inside out list to be able to print total number of files visited in -stats option.

#BEFORE

This option find all the files that are last modified before a given date or a given datetime inside the out list. It checks if the given date or datetime consist the character ‘T’ or not. If it consists the character ‘T’ then convert the given time into necessary datetime format and check the files one by one. If it does not consist the character ‘T’ convert the given time into necessary date format and check the files one by one. It uses *it_is_before(x)* function to check the files. The implementation is as follows:

```
if args.before != "None":
        if ("T" in args.before):
                mod_before = time.mktime(datetime.datetime.strptime(args.before, "%Y%m%dT%H
%M%S").timetuple())
        else:
                parse_str(args.before)
                mod_before = time.mktime(datetime.datetime.strptime(args.before, "%Y%m
%d").timetuple())
        parsed_before = parse_str(mod_before)
        def it_is_before(x):
                mod_x = os.path.getmtime(x)
                parsed_x = parse_str(mod_x)
                if parsed_x < parsed_before:
                        return True
                return False
        out[:] = [x for x in out if it_is_before(x)]
```

#AFTER

This option find all the files that are last modified after a given date or a given datetime inside the out list. It checks if the given date or datetime consist the character ‘T’ or not. If it consists the character ‘T’ then convert the given time into necessary datetime format and check the files one by one. If it does not consist the character ‘T’ convert the given time into necessary date format and check the files one by one. It uses *it_is_after(x)* function to check the files. The implementation is as follows:

```
if args.after != "None":
        if ("T" in args.after):
                mod_after = time.mktime(datetime.datetime.strptime(args.after, "%Y%m%dT%H
%M%S").timetuple())
        else:
                parse_str(args.after)
                mod_after = time.mktime(datetime.datetime.strptime(args.after, "%Y%m
%d").timetuple())
        parsed_after = parse_str(mod_after)
        def it_is_after(x):
                mod_x = os.path.getmtime(x)
                parsed_x = parse_str(mod_x)
        if parsed_x > parsed_after:
                return True
        return False
        out[:] = [x for x in out if it_is_after(x)]
```

#MATCH

This option finds all the files that have names which matches a given python regular expression. It uses a for block to check all the file names inside the out list. The implementation is as follows:

```
if args.match != "None":
        new_out = []
        new_out.extend(out)

        for x in new_out:
                pattern = re.compile(args.match)
                found = pattern.match(os.path.basename(x))
                if not found:
                        out.remove(x)
```

#BIGGER

This option finds all the files that have sizes bigger than or equal to the given size. It uses a for loop and an if block to perform this procedure. First it formats the given size according to it's size type (kilobytes, megabytes, gigabytes) inside the if block, then it checks every file inside the out list by using *it_is_bigger(x)* function. The implementation is as follows:

```
def it_is_bigger(x):
        filesize = os.path.getsize(x)
        if filesize > parsed_size:
                return True
        return False

if args.bigger != "None":
        size_in_args = ""
        for x in args.bigger:
                if(x != 'G' and x != 'M' and x != 'K'):
                        size_in_args = size_in_args + x
        x = args.bigger[-1]

        parsed_size = parse_str(size_in_args)

        if(x == 'G'):
                parsed_size = parsed_size * pow(10,9)
        elif (x == 'M'):
                parsed_size = parsed_size * pow(10,6)
        elif (x == 'K'):
                parsed_size = parsed_size * pow(10,3)
        else:
                parsed_size = parsed_size

        out[:] = [x for x in out if it_is_bigger(x)]
```

#SMALLER

This option finds all the files that have sizes smaller than or equal to the given size. It uses a for loop and an if block to perform this procedure. First it formats the given size according to it's size type (kilobytes, megabytes, gigabytes) inside the if block, then it checks every file inside the out list by using *it_is_smaller(x)* function. The implementation is as follows:

```
def it_is_smaller(x):
        filesize = os.path.getsize(x)
        if filesize < parsed_size:
                return True
        return False
if args.smaller != "None":
        size_in_args = ""
        for x in args.smaller:
                if(x != 'G' and x != 'M' and x != 'K'):
                        size_in_args = size_in_args + x
        x = args.bigger[-1]
        parsed_size = parse_str(size_in_args)
        if(x == 'G'):
                parsed_size = parsed_size * pow(10,9)
        elif (x == 'M'):
                parsed_size = parsed_size * pow(10,6)
        elif (x == 'K'): #if x == K
                parsed_size = parsed_size * pow(10,3)
        else:
                parsed_size = parsed_size
        out[:] = [x for x in out if it_is_smaller(x)]
```

#ZIP

This option packs all the files inside the given or current directory as a zip file. It checks if the out list is empty or not. If it is not empty, then it performs the packing procedure. The implementation is as follows:

```
if args.zip != "None":
        if out:
                command = "zip " + args.zip + " "
                for filepath in out:
                        command = command + filepath + " "
                os.popen(command).read()
```

#DELETE

This option deletes all the files inside the out list. The implementation is as follows:

```
if args.delete == True:
        if out:
                command = "rm "
                for filepath in out:
                        command = command + filepath + " "
                os.popen(command).read()
```

#DUPLCONT

This option finds all the files that have the same content in sorted order. It uses *shasum* command to hash all the files inside the out list. Then it appends every file and it's hash value to *duplcont_dict* dictionary and checks the hash values if they are the same or not after sorting the dictionary. The implementation is as follows:

```
duplcont_dict = {}
if args.duplcont == True:
        for filepath in out:
                command = "shasum " +filepath
                command_out = os.popen(command).read()
                hash_out = ""
                for x in command_out:
                        if(x != ' '):
                                hash_out += x
                        else:
                                break
                if hash_out in duplcont_dict:
                        duplcont_dict[hash_out].append(filepath)
                else:
                        duplcont_dict.setdefault(hash_out,[filepath])
                duplcont_dict[hash_out].sort()
```

#DUPLNAME

This option finds all the files that have the same name in sorted order. It appends every file name and it's path to *duplname_dict* dictionary. In this way, it finds all the files that have the same names inside the *duplname_dict* dictionary. The implementation is as follows:

```
duplname_dict = {}
if args.duplname == True:
        for filepath in out:
                if os.path.basename(filepath) in duplname_dict:
                        duplname_dict[os.path.basename(filepath)].append(filepath)
                else:
                        duplname_dict.setdefault(os.path.basename(filepath),[filepath])
        duplname_dict = collections.OrderedDict(sorted(duplname_dict.items()))
```

#STATS

This option finds traversal statistics to print these statistics at the end of the program. It calculates total number of files visited and total size of files visited at the beginning of the program and it calculates total number of files listed and total size of files listed at the end of the program. The implementation is as follows:

```
#Beginning of the program (after doc_list)
files_visited = len(out)
sizes_visited = 0
for x in out:
        filesize = os.path.getsize(x)
        sizes_visited += filesize
```

```
#End of the program
files_listed = len(out)
sizes_listed = 0
for x in out:
        filesize = os.path.getsize(x)
        sizes_listed += filesize
unique_files = 0
unique_sizes = 0
if args.duplcont == True:
        unique_files = len(duplcont_dict)
        for x in duplcont_dict:
                unique_sizes += os.path.getsize(duplcont_dict[x][0])
elif args.duplname == True:
        unique_files = len(duplname_dict)
```

#PRINTING OUT LIST

Program checks the command-line command if it contains the **-nofilelist** command or not. If it does not contain the option, then it prints the files that remain in the out list at the end of the program. If it contains the **-nofilelist** option, program will not produce any output. The implementation of printing part is as follows with it's explanation in more detail:

```
if args.nofilelist == 'None':                                   #if -nofilelist option is not select
        if args.duplcont == True:                                       #if -duplcont option is given
                for x in duplcont_dict:                         #prints the contents of duplcont dictionary
                        print "-----"
                        if len(duplcont_dict[x]) > 1:
                                for y in duplcont_dict[x]:
                                        print y
                        else:
                                print duplcont_dict[x][0]
                print "-----"
        elif args.duplname == True:                                     #if -duplname option is given
                for x in duplname_dict:                         #prints the contents of duplname dictionary
                        print "-----"
                        if len(duplname_dict[x]) > 1:
                                for y in duplname_dict[x]:
                                        print y
                        else:
                                print duplname_dict[x][0]
                print "-----"
        else:                                   #if no extra option is given, prints the contents of out list
                for x in out:
                        print x

if args.stats == True:                  #if -stats option is given then prints the additional informations

        print "number of files visited: " + str(files_visited)
        print "total size of files visited: " + str(sizes_visited)
        print "number of files listed: " + str(files_listed)
        print "total size of files listed: " + str(sizes_listed)
#below lines will be executed if -stats and -duplcont or -duplname options are given together
```

```
    if args.duplcont == True:
        print "number of unique files listed: " + str(unique_files)
        print "total size of unique files listed: " + str(unique_sizes)
    elif args.duplname == True:
        print "number of unique files listed: " + str(unique_files)
```

HOW TO RUN

→First, open a new terminal
→Then, go to filelist.py's directory
→Then, run the program by typing "***python filelist.py [options] [directory list]***"
    →options and directory list are optional